On the Need for Autonomic Connection Management to Speed up WSRF-Enabled Web Service Calls

Michael Welzl, Muhammad Murtaza Yousaf, and Alex Villazon

Institute of Computer Science, Distributed and Parallel Systems Group
University of Innsbruck, A-6020 Innsbruck, Austria
{Michael.Welzl,Murtaza.Yousaf,Alex.Villazon}@uibk.ac.at

Abstract. The network stack that comes into play when a Web Service is called is quite large; the fact that the notion of state is not properly communicated through the stack leads to an unnecessary performance deterioration that can be severe for Grid applications which consist of computationally small parts. This fact, which is due to TCP connection setup and teardown delay, therefore limits the parallelization granularity that is feasible in a Grid — it requires special attention when applying Web Service concepts such as orchestration. We explain the problem, describe its occurrence in a simple experiment, discuss some implications and outline some architectural design choices that could allow to bypass this seemingly inevitable communication overhead.

Key words: Grid, Globus, WSRF, Granularity, TCP, Connection

1 Introduction

Abstraction is a powerful concept. In the context of computer networks, it is realized in the form of a layer stack, where each layer utilizes the layer directly underneath it to provide a service to the layer above; details of the service implementation are hidden within each layer. This stack makes it possible to realize arbitrarily complex architectures — such as a Grid — on top of any network without taking network specific issues like routing, medium access or congestion control into account. There is, however, a certain danger of losing efficiency with each abstraction step unless it is carried out with special care. We argue that the parallelization granularity of Grid applications is unnecessarily limited, and that this potential performance problem is caused by the historic evolution of the related Grid and network layers.

While this may not be very important for today's typical Grid applications, which consist of computationally intensive parts that may only communicate sporadically except for occasional bulk data transfer, it could become increasingly relevant in the near future: since the research areas of Grid computing and Web Services have converged, it appears to be a natural choice to apply Web Service concepts such as orchestration (where a Web Service is built from

distributed atomic low-level functions) to a Grid, e.g., in the form of business process composition in a workflow.

This paper is organized as follows: In the next section, we will briefly analyze what happens during a WSRF-enabled Web Service call at the transport layer and above, thereby identifying the performance limitation that is due to TCP connection setup and teardown. An architectural direction for coping with this problem by autonomically handling connection information within the stack is described in section 3; we provide an overview of related work in section 4 and conclude in section 5.

2 Problem Analysis

Here is a rough overview of what happens when a remote WSRF-enabled Web Service is called with the Globus Toolkit version 4.0 [1], assuming that a deployed service has already started in a hosting environment:

- The information is converted into an XML description, which is handed over to SOAP.
- SOAP utilizes HTTP to transfer the data. HTTP provides a reliable message oriented text based communication service.
- HTTP uses TCP, which provides the notion of a reliable FIFO pipe from sender to receiver.
- Up to now, the information has merely been converted to text and enriched with protocol specific information (headers). TCP also adds a header, but additionally, it requires some messages to go back and forth in order to provide a connection oriented service on top of the IP protocol's unreliable datagram transmission:
  - A connection is established using a three-way handshake.
  - The actual communication is carried out; for each transmitted data segment, an acknowledgment is generated by the receiver.
  - The connection is closed using yet another handshake from either side.
- IP adds a header and sends the TCP segments, which may (but should not be) split into several fragments. All kinds of things happen to IP packets as they traverse the Internet, but they are then under the control of routers — typically, this is beyond the influence of end hosts belonging to a Grid.
- Data link layer protocols also add headers and carry out some extra work such as buffering and retransmission in case of transmission errors. At this point, we decide to neglect everything underneath IP because it depends on the specific local infrastructure while the Grid should work across the global Internet.

There are several places in this list where things may become inefficient: SOAP handling, for one, is text based, which introduces some overhead depending on local processing power and message length. A detailed analysis was already carried out by other authors (see section 4). HTTP can mainly introduce overhead by requiring several TCP connections for a single communication

flow, but this problem was alleviated by the introduction of "persistent connections" in HTTP 1.1 [2]. IP routers and data link layer mechanisms are typically not under control of a Grid application or middleware; this leaves us with TCP overhead which, in our opinion, deserves a closer look.

Any sort of remote procedure call can be classified as request/response oriented communication. As such, it requires at least the following basic messages to be transferred:

- *Request:* "please execute X (with parameters Y)"
- *Response:* "done, here is the result"

If such a communication utilizes TCP, at least nine messages have to be transferred: a three-way handshake that opens the connection, followed by the actual request and response messages and four messages that are required to close the connection (assuming that messages are interspersed so that there are no extra acknowledgments required for the request and reply packets). This leads to a delay of at least three round-trip times (RTTs) at either end of the connection.

As we will see, connections are opened and closed for each consecutive call to a remote Web Service, even if the remote machine stays the same and the calls are carried out one after another. Furthermore, when utilizing a WS-resource for the first time, an instance may need to be created using a Factory. The aforementioned request is then preceded by the message that requests creation of the resource, causing at least two more TCP connections to be opened and closed in succession. Clearly, this is inefficient in a widely distributed Grid with long delays.

2.1 Traces

We carried out a short test in order to examine the real-life behavior of the popular Globus Toolkit 4.0; since we wanted to show the best case scenario (minimum required communication overhead), we started a non secure (HTTP) container, which means that only transport security was disabled and message security could still be used. Specifically, the following tasks were executed and traced with "ethereal" [3] between local machines A and B after starting the container at B with "globus-start-container":

1. Resource creation at B, spawned from A
2. Web Service calls from A to B

Our Web Service merely realized a "Hello world" type of communication. We used CounterService [4] for our analysis in which we first create a resource (counter) that has some default value. Afterwards, with each service call we pass a number, the counter is increased by the passed argument and the increased counter value is returned. The trace from the resource creation consists of the packets shown in table 1. Table 2 shows the exchange of packets for the Web Service call to increase the counter. We started and called this service on three

Table 1. Resource creation at B, spawned from A

Packet No.	*Source*	*Destination*	*Information*	*Protocol*
1	A	B	SYN	TCP
2	B	A	SYN, ACK	TCP
3	A	B	ACK	TCP
4	A	B	*POST/wsrf/service*	*HTTP*
5	B	A	ACK	TCP
6	A	B	*Cont.*	*HTTP*
7	B	A	ACK	TCP
8	A	B	*soapenv*	*HTTP*
9	B	A	ACK	TCP
10	B	A	*HTTP/1.1*	*HTTP*
11	A	B	ACK	TCP
12	B	A	*Cont.*	*HTTP*
13	A	B	ACK	TCP
14	B	A	*Cont.*, FIN	*HTTP*
15	A	B	ACK	TCP
16	A	B	FIN, ACK	TCP
17	B	A	ACK	TCP

different machines many times in a row and noticed that traces were identical every time.

Since tables 1 and 2 both show that a TCP connection was opened (the SYN packets) and closed (the FIN packets), merely creating a resource and calling a service required two TCP connections to be opened and closed. According to the description above, this accounts for at least four round-trip times. Calling a service right after resource creation required another connection setup/teardown combination (another two round-trip times), as is the case for a subsequent call to the same Web Service — connections are not reused. A single call to a Web Service which must have a resource, before it can be used, therefore accounts for a total of at least 30 packets being transmitted and takes at least 6 round-trip times.

Let us take a closer look at the procedures shown in table 1 and find out why there are more than the 9 packets that we mentioned earlier as an ideal case: first, the three-way handshake is carried out (the three SYN, SYN/ACK and ACK packets). Then, the request is transmitted from A to B via SOAP in the form of an HTTP POST message (HTTP related information is shown in italics in tables 1 and 2). These four packets concur with our earlier description. The remaining five packets that are required even in an ideal scenario are the HTTP response (packet number 10) and the four last packets (numbers 14 to 17). The "inefficient" packets number 5, 7, 9, 11, and 13 stem from layering: since the TCP instance in the TCP/IP stack of B cannot know when the local HTTP instance will answer, it sends the acknowledgment right away and does not piggyback it onto the HTTP response. Packets number 6, 8, and 12 are caused by the HTTP messages being divided into several packets. This is not

Table 2. Web Service call from A to B

Packet No.	*Source*	*Destn.*	*Information*	*Protocol*
1	A	B	SYN	TCP
2	B	A	SYN, ACK	TCP
3	A	B	ACK	TCP
4	A	B	*POST/wsrf/service*	*HTTP*
5	B	A	ACK	TCP
6	A	B	*Cont.*	*HTTP*
7	B	A	ACK	TCP
8	A	B	*Cont.*	*HTTP*
9	B	A	ACK	TCP
10	A	B	*soapenv*	*HTTP*
11	B	A	ACK	TCP
12	A	B	*Cont.*	HTTP
13	B	A	ACK	TCP
14	B	A	*HTTP/1.1*	*HTTP*
15	A	B	ACK	TCP
16	B	A	*Cont.*	*HTTP*
17	A	B	ACK	TCP
18	B	A	TCP	FIN, ACK
19	A	B	TCP	ACK
20	A	B	TCP	FIN, ACK
21	B	A	TCP	ACK

ideal because, in our case, the total size of all these packets never exceeded the "Maximum Transfer Unit" (MTU) of the link between A and B.[1]

Table 2 only differs from table 1 in that more HTTP packets were required to transmit the messages. It might have been possible to merge the two consecutive connections that were required for resource creation and service call into one in theory. Further, seperate connections were used for two consecutive service calls which could also be possible in a single connection because the HTTP version implemented in GT4 is 1.1, which supports persistent connections.

2.2 Implications

The additional delay due to TCP overhead may lead to problems if the parallelism granularity in a Grid application is small (e.g., when distributing short tasks within a loop or composing a Web Service from small atomic parts) and RTTs are large. In what follows, we underpin our arguments with a simple quantitative example: we concentrate on the client side and assume a simple request/response oriented communication. Furthermore, we assume a simplified model where the total duration at the client is the sum of the TCP related messages (roughly 2 RTTs) and the time of the execution of the Web service

[1] The MTU of a link is the length of the largest packet that can be transmitted without being fragmented.

(computation). In figure 1 we plot the RTT overhead as a percentage of the whole time for the request/response communication, e.g. $\frac{2*RTT}{2*RTT+Computation}$, against increasing computation time (in milliseconds). Furthermore, we also plot $\frac{RTT}{RTT+Computation}$ — we will comment on this in the following section.

¿From this simple plot we can deduce that TCP overhead is non neglible for small computation times, but it vanishes with increasing computation time. This reduction is less drastic for large RTT values. The implications based on these results are:

- Grid applications should avoid computationally small remote Web Services (if this is possible). Furthermore, any sort of fine grain parallel computations realized through Web Services will be affected by this performance problem.
- Grid applications need fast and accurate performance information (performance predictions) about network conditions (like RTT) to remote computers where the needed Web Services are located. If a Grid application can choose among different computers offering the needed Web Service, an accurate performance information can be a big help when optimizing response time. This is especially true for applications relying on many computationally small Grid services.
- Grid applications need fast and accurate performance estimates about Web Service run-times. From such estimates the applications can deduce the potential overhead that TCP could cause.

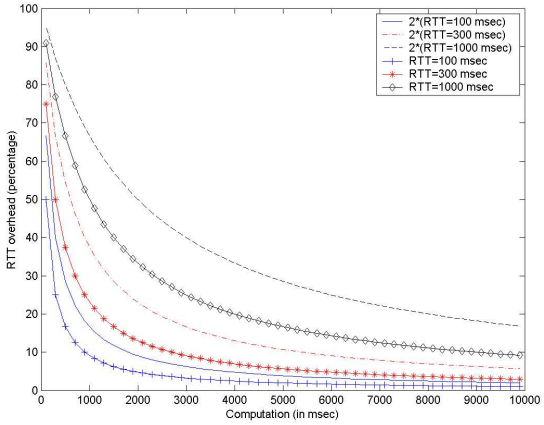

Fig. 1. Percentage of RTT overhead with increasing computation time

3 Proposed Solution

A web server can use the persistent connections feature of HTTP 1.1 because the elements of a web page logically belong together. The SOAP specification does not make prescriptions regarding stateful or stateless operation. Web Services, however, were originally designed to be stateless, i.e. there was no use for persistent connections. With the stateful extensions in WSRF, persistent connections again make sense, but due the historic evolution of layering in the Web Services/SOAP/HTTP/TCP/IP stack, the notion of a connection is not available at the WSRF level.

Logically, the concepts of "state" and "persistent connections" belong together; control over persistent connections must therefore be executed at a layer where the concept of state exists, and then communicated through the stack (i.e. it must be ensured that the notification "maintain a persistent connection" or "tear down a persistent connection" is handed over until it reaches the HTTP level).

One possible solution would be to involve the user (in this case, the Grid programmer). Indeed, the programmer of a Grid application has the most precise knowledge of and when a WS-Resource will be used, and how often it will be accessed again, as the control flow of the Grid application depends on various things that can clearly not be known by the Web Service, WSRF, SOAP and HTTP implementations. Involving the programmer could take the form of special "start session" and "end session" function calls that indicate for how long it would make sense to maintain a connection.

However, in Grid Computing, it is a major research trend to *simplify* the (already daunting) programming effort, and handle as many functions as possible in an autonomic fashion. Thus, such design which would put an additional burden onto the Grid programmer is at odds with the general trend and would not be well received.

We believe that TCP connections should be handled in an *autonomous* fashion, where the Grid middleware automatically detects whether to keep a connection open or close it. Then, it is a tricky question to figure out the ideal location for this change: should SOAP be changed, or the Web Service implementation, or WSRF — and is it a client or server side change, or both?

Our answer is: since all state related features are part of the WSRF implementation, this is where control over persistent connections should be executed. From tables 2 and 1, we can see that, as with any other HTTP request, tearing down the TCP connection is initiated on the server side. This is where the decision to keep a connection open would have to be made. Clearly, we cannot simply keep connections open forever, but it would be feasible to maintain a fixed size cache of connections. The only information to be stored in this cache is the "Transmission Control Block (TCB)", a set of data such as port and sequence numbers that identify the current state of a TCP connection. Then, several cache maintenance strategies could be implemented:

– Connections can be added to the cache when...

  - A resource was created. This can be identified at the server side because the server must include Reference Properties for addressing the resource in the response to the call that led to the resource creation.
  - A resource is used (Reference Properties are included in a call from the client).
  - The WS-Base Notification feature of WSRF is used such that the server acts as a NotificationProducer (which informs a NotificationConsumer about events related to a specific resource) and a subscribe request message arrives.
- Connections can be removed from the cache when...
  - a per-connection timer expires; this timer should be refreshed whenever a connection is used again.
  - the cache grows beyond a certain limit.
  - a resource is destroyed (explicitly or because its lifetime is exceeded — this is part of the WS-Lifetime feature of WSRF)

4 Related Work

The efficiency of SOAP for high performance communication was mainly analyzed from a local processing perspective in [7] and [8]; these efforts complement our work. This is particularly true for reference [9], which contains a description of the actual SOAP communication overhead. We would like to point out that, despite indications that local SOAP processing can account for a larger amount of delay than TCP connection setup and teardown, SOAP processing is likely to become faster as a natural result of Moore's law. TCP connection handling, however, is a fixed unnecessary overhead that has natural lower limits such as the inevitable physical delay for transferring a signal across a satellite link (in case of a massively distributed Grid).

In accordance with our initial observations, layering "remote procedure call" type of communication on top of HTTP was identified as a potentially poor choice in [10]. On the transport level, it would theoretically even be possible to reduce the minimal 9 packets (3 round-trip times) overhead: In [5], this process is outlined in comparison with an experimental TCP extension entitled "Transactional TCP (T/TCP)", where the connection setup, request, teardown and response messages are piggybacked in a way that reduces the total number of messages to three, leading to a total delay of only one round-trip time at either end of the connection. The implication of such a reduction can be seen in figure 1: for a given RTT, the overhead is reduced by a certain amount. This reduction is more pronounced for larger RTT values. Therefore, such an alternative could be a valuable performance boost for Grid applications which utilize computationally small Grid services or have large RTTs.

However, T/TCP, which is specified in [6] and designed for any kind of transactional communication exchange, can be expected to remain an experimental specification because of security drawbacks. As one example, one would not want

to allow an unauthenticated source to execute a local method — but authentication requires some additional communication, which appears to render T/TCP useless.

The problem described in this document applies to WSRF-enabled Web Services of any kind; however, we believe that the issue is most important in the Grid context because the Grid is meant to support high performance distributed computing — performance may be less relevant in typical Web Services usage scenarios. We therefore contributed a brief overview of the problem to [11], which is a working document of the Global Grid Forum "Grid High Performance Networking" research group; the goal of this document is to point out such problems to the Grid community.

5 Conclusion and Future Work

Several other factors appear to make TCP less than ideal for transmission of function calls — its embedded congestion control mechanism, for instance, assumes a certain connection duration. It starts with only one segment per round-trip time and may not even reach its "Congestion Avoidance" state before the end of the communication [12]. We therefore intend to extend the work described in this document with an analysis of more exhaustive Web Service calls (e.g., calls with several more complex parameters such as arrays) in the future. Based on this analysis, we will implement our solution which autonomically manages TCP connections on the WSRF level.

The traditional notion of a "Grid Service" has recently been refactored towards Web Service technology. The Web Service Resource Framework (WS-RF) [14] has replaced the Open Grid Service Infrastructure (OGSI), which GT3 is based upon. Yet, despite this significant evolution, TCP connection management did not change: we already examined the network overhead of GT3 Grid Service calls in an earlier study [11] and obtained results equal to the ones presented in this paper. This leads us to conclude that the only way to enhance the situation is to implement an autonomic connection management scheme such as the one described in this paper.

Acknowledgments. We would like to thank Stefan Podlipnig and Gregor Mair for their contributions.

References

1. The Globus toolkit: http://www-unix.globus.org/toolkit/
2. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. *RFC 2616*.
3. ethereal : http://www.ethereal.com/
4. Counter Service Example : http://www.globus.org/toolkit/docs/4.0/common/javawscore/Java_WS_Core_Samples.html#s-javawscore-Core_Samples-counter

5. Tanenbaum, A. (2003). *Computer Networks (fourth edition).* Pearson Education, New Jersey.
6. Braden, R. (1994). T/TCP – TCP Extensions for Transactions Functional Specification. *RFC 1644.*
7. Chiu, K., Govindaraju, M., Bramley, R. (2002). Investigating the Limits of SOAP Performance for Scientific Computing. *Proceedings of High Performance Distributed Computing (HPDC).* Edinburgh, Scotland.
8. Kohlhoff, C., Steele, R. (2003). Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. *Proceedings of WWW2003.* Budapest, Hungary.
9. Davis, D., Parashar, M. (2002). Latency Performance of SOAP Implementations. *Proceedings of Cluster Computing and the Grid (CCGrid).* Berlin, Germany.
10. Moore, K. (2002). On the use of HTTP as a Substrate. *RFC 3205.*
11. Volker Sander (ed.), William Allcock, Pham CongDuc, Jon Crowcroft, Mark Gaynor, Doan B. Hoang, Inder Monga, Pradeep Padala, Marco Tana, Franco Travostino, Pascal Vicat-Blanc Primet, Michael Welzl: "Networking Issues for Grid Infrastructures", *Global Grid Forum Document GFD.37 (informational), Grid High Performance Networking Research Group, 22 November 2004.*
12. Fredj, S. B., Bonald, T., Proutiere, A., Regnie, G., Roberts, J. W. (2001). Statistical Bandwidth Sharing: A Study of Congestion at Flow Level. *Proceedings of ACM SIGCOMM.* pp. 111 - 122.
13. tcpdump: http://www.tcpdump.org/
14. WS-RF: http://www.globus.org/wsrf