Performance von XML-RPC

Ralf Döring
Technische Universität Ilmenau
ralf.doering@prakinf.tu-ilmenau.de

Abstract. Auf XML basierende RPC Mechanismen stellen einen praktikablen Ansatz zur Nutzung von Diensten in heterogenen Systemen dar. Durch die aufwendige Kodierung der zu übertragenden Informationen ist jedoch mit Leistungsengpässen zu rechnen. Der folgende Beitrag zeigt anhand des Beispiels von XML-RPC einige performancekritische Schwachpunkte auf. Die getroffenen Aussagen sind sinngemäss auch auf andere XML basierende RPC Systeme wie SOAP übertragbar.

1 Einleitung

Das Client/Server Konzept stellt heute die Grundlage für Systeme zur verteilten Ver- und Bearbeitung von Daten dar. Dabei bieten diese Systeme in der Regel Schnittstellen an, die auf dem Paradigma des *Remote Procedure Call (RPC)* aufbauen. Trotz der langen Tradition dieser Technologien sind einfach zu nutzende und zwischen verschiedenen Systemen einsetzbare Realisierungen selten. Zwar stehen für verschiedene Technologien wie CORBA, ONC-RPC und DCOM entsprechende Werkzeuge bereit, Systeme die gleichzeitig Schnittstellen für verschiedene RPC-Realisierungen anbieten existieren aufgrund der großen Unterschiede in den genannten Basistechnologien jedoch kaum.

Als Ausweg aus diesem Dilemma bietet sich die Verwendung eines Ansatzes an, welcher ohne großen Aufwand in eigenen Systemen implementiert werden kann. Als Lösung bieten sich dabei auf XML basierende RPC Mechanismen förmlich an, da für alle gängigen Entwicklungssysteme eine Unterstützung für die Verarbeitung von XML Dokumenten heute als gegeben vorausgesetzt werden kann. Die Abbildung von RPC Mechanismen kann dann entweder selbst realisiert oder unter Zuhilfenahme existierender Lösungen wie XML-RPC [1] oder SOAP [2][3] implementiert werden. Anhand von XML-RPC soll im folgenden an einigen Beispielen gezeigt werden, mit welchen Problemen gerechnet werden muss.

1.1 XML-RPC

XML-RPC ist eine relativ simple Spezifikation zur Übertragung von Remote Procedure Calls mittels HTTP. Es existieren Bibliotheken für die Anbindung an alle gängigen Programmiersprachen und Entwicklungssysteme. Ein XML-RPC Request wird mittels eines HTTP POST Requests(vgl. dazu [4]) übertragen. Der eigentliche RPC wird als XML-kodierter Entitiebody im POST-Request übetragen.

```
<?xml version='1.0'?>
<methodCall>
   <methodName>ping</methodName>
   <params></params>
</methodCall>
```

Abbildung 1. Einfacher XML-RPC Aufruf

Ein einfaches Beispiel für einen XML-RPC Aufruf zeigt Abbildung 1. Hier wird die XML-RPC Methode `ping` ohne zusätzliche Parameter aufgerufen. Für das genaue Verständnis und die Spezifikation der XML-RPC Request- und Responseformate sei auf die unter [1] zu findende Spezifikation verwiesen.

1.2 Testumgebung

Als Testumgebung kamen zwei Linux-PCs zum Einsatz, welche über ein vollgeswitchtes 100 MBit Netzwerk miteinander verbunden waren. Diese Realisierung ermöglicht die Nutzung der vollen zur Verfügung stehenden Bandbreite. Den jeweiligen Systemen wurde eine eindeutige Rolle als XML-RPC Server bzw. XML-RPC Client zugewiesen, die während der Tests nicht verändert wurden.

Der XML-RPC Server stellt für die Tests drei XML-RPC Routinen bereit, welche die in Abschnitt 2 vorgestellten Testfälle abdecken. Die Routinen bieten dabei folgende Funktionalität:

- Einfacher XML-RPC Methodenaufruf ohne Transfer von Methodenparametern.
- Methodenaufruf mit Übermittlung eines Integer-Arrays variabler Länge.
- Methodenaufruf mit Übermittlung eines Strings variabler Länge.

Alle Routinen übermitteln als Ergebnis einen Integerwert an den Aufrufer zurück.

Die Software der Testumgebung besteht aus zwei Teilen: dem XML-RPC Server und dem XML-RPC Client. Ursprünglich existierten dazu jeweils Implementierungen in Python (Version 2.0) , die auf der XML-RPC Bibliothek `xmlrpclib.py`[5] basierten. Die Testverfahren sind so ausgelegt, dass der Großteil an Verarbeitungsleistung auf Seiten des Servers erbracht wird. Um die Ergebnisse ohne den Kontext der Implementierungssprache betrachten zu können, existiert eine zweite Realisierung des XML-RPC Servers in Java, welche die gleiche Funktionalität zur Verfügung stellt. Dieser Server wurde mit dem JDK 1.3.0 sowie der *„HELMA XML-RPC Bibliothek für Java“*[6] realisiert. Der „Serverpart“ wurde von einem mit Intel Pentium III Prozessor mit 850 MHz ausgerüsteten System übernommen, die Rolle des Clients übernahm ein System mit AMD Athlon 750. Beide System sind mit 256MB Speicher ausgestattet, laufen unter Debian/GNU Linux und lagerten zum Testzeitpunkt keinen Speicher auf externe Medien aus.

2 Performancetests

Die einzelnen Tests fanden in der oben beschriebenen Testumgebung statt. Die Diagramme bestehen aus einzelnen Messpunkten, die zu besseren Visualisierung von Trends und Verhältnissen durch Linien verbunden worden sind. Die einzelnen Messpunkte sind das Mittel der Laufzeiten für den jeweiligen XML-RPC Aufruf und wurden durch Zeitmessung über eine größere Anzahl von Aufrufen und anschließender Mittelwertbildung gewonnen. Die gewonnenen Mittelwerte aus diesem Messverfahren sind aufgrund der Einschränkungen der einfachen Messmethode ohne Betrachtung von Randbedingungen in ihren Absolutwerten nicht tauglich für konkrete Aussagen hinsichtlich genauer Perfomanceparemeter der eingesetzten Technologien. Da im Mittelpunkt der Betrachtungen die Abschätzung von Größenordnungen der Performanceverluste der XML Mechanismen gegenüber herkömmlichen RPC-Verfahren stehen soll, scheint diese Einschränkung annehmbar zu sein.

Die einzelnen Tests sind in ihrer Natur künstlich und spiegeln das reale Verhalten von Anwendungen nur schlecht wieder. Aus diesem Grund sind Messungen anhand des Applicationservers *Zope* im Gange, welcher neben HTTP auch den Zugriff mittels XML-RPC unterstützt. Anhand dieser Messungen ist eine Aussage über die Verhältnisse beim Einsatz von XML-RPC in Anwendungsumgebungen möglich.

2.1 Test 1: Ping

Realisierung	Zeit für Aufruf (in Sekunden)	Requests/s
Java RMI	0.000623	1605
Server Java	0.005988	167
Server Python	0.007134	140

Tabelle 1. Methodenaufruf ohne Daten

Als erster Test erfolgt die Betrachtung einzelner Methodenaufrufe. Die zugehörige XML-RPC Methode übernimmt dabei keine weiteren Parameter und beantwortet den Request wie in allen anderen Testfällen durch Übermittlung einer Integerkonstante. Als Vergleichswert wurden die Ergebnisse einer äquivalenten Anwendung unter Nutzung von RMI mit in die Ergebnisliste in Tabelle 1 aufgenommen.

2.1.1 Interpretation der Ergebnisse

Tabelle 1 zeigt das arithmetische Mittel über jeweils 10000 Methodenaufrufe. Wie in allen anderen Testfällen wurde eine XML-RPC Serverimplementierung in Java sowie eine Implementierung in Python genutzt. Die Ergebnisse scheinen mit Aufrufzeiten von rund 6 `ms` für die Javaimplementierung und etwa 7 `ms` für die Pythonimplementierung recht passabel zu sein, weisen jedoch ein um circa den Faktor 10 schlechteres Laufzeitverhalten als eine Implementierung mittels Java-RMI auf. Eine zum Vergleich hinzugezogene Messung von Prozeduraufrufen anhand einer analogen Implementierung mittels SUN-RPC [7][8] ergab einen Vergleichswert von 0.13 `ms`.

Es ist somit zu erkennen, dass der Overhead durch die Kodierung und Dekodierung des eigentlichen RPC in eine XML Beschreibung bereits einen nicht zu vernachlässigenden Anteil an Rechenleistung benötigt. Während der Abarbeitung des Testlaufs lag die CPU-Auslastung des ansonsten unbelasteten Serversystems bei 100%, so dass als begrenzende Ressource in obigen Testfällen nicht das Netzwerk sondern die Rechenleistung des Testsystemes angesehen werden kann.

2.2 Test 2: Transfer von Integerlisten

Mit diesem Test soll das Verhalten bei der Übertragung von Integerarrays unterschiedlicher Größe untersucht werden. Der Client sendet in diesem Szenario Listen mit den im Diagramm in Abbildung 2 angegebenen Längen, die jeweils mit den fortlaufenden Integern von 1 bis zur Listenlänge besetzt sind. Als Vergleichswert ist im Diagramm zusätzlich das Laufzeitverhalten einer äquivalenten Implementierung auf Basis von RMI eingefügt.

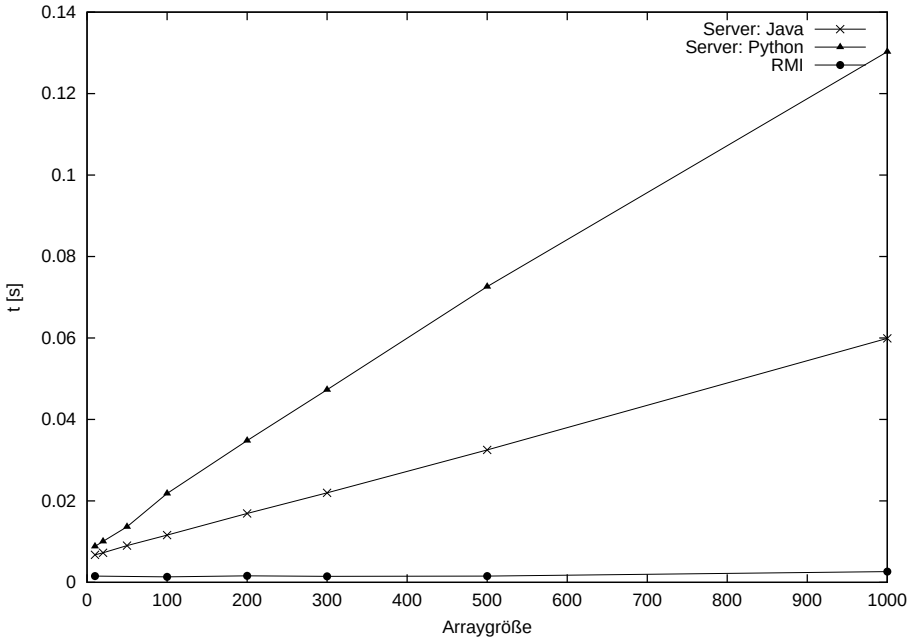


Abbildung 2. Ping mit Integerlisten

Auch wenn dieser Test sehr künstlich wirkt, lassen sich so wertvolle Erkenntnisse über die Effizienz der Kodierung von Arrays mittels XML-RPC gewinnen.

2.2.1 Interpretation der Ergebnisse

Die Zeiten für die Abarbeitung der Requests steigen linear mit der Größe der übertragenen Listen an. Da auf Seiten des Servers jedoch keinerlei Verarbeitung der übertragenen

Daten stattfindet, bedarf der Unterschied in den Ergebnissen bei Verwendung des Java bzw. Pythonservers genauerer Erklärung. Die Erklärung ist jedoch simpel: Der starke Geschwindigkeitsvorteil der Javaimplementierung liegt einzig in der Verwendung des JIT-Compilers. Wird dieser deaktiviert, so ist die javabasierte Implementierung des Servers wiederum langsamer als die Pythonimplementierung.

Genauere Betrachtung verdient jedoch der Umstand, das ein Methodenaufruf selbst mit verhältnismäßig kleinen Datenmengen in Form von Arrays um Größenklassen langsamer als ein einfacher Methodenaufruf ohne Datenübermittlung. Die Unterschiede belaufen sich hier zum Beispiel auf Faktor 10 für den Pythonserver bzw. Faktor 5 für den Javaserver bei Betrachtung einer Listengröße von 500 Elementen.

Diese Diskrepanz liegt in der Parameterkodierung beim Transfer von Arrays mittels XML-RPC begründet. Durch die explizite Typisierung des übertragenen Datenstroms ist eine aufwendige Dekodierung notwendig, die mittels kostenintensiver Stringvergleiche erfolgen muss.

```
<?xml version='1.0'?>
<methodCall>
  <methodName>ping</methodName>
   <params>
   <param><value><array><data>
      <value><int>1</int></value>
      <value><int>2</int></value>
      <value><int>3</int></value>
      <value><int>4</int></value>
      <value><int>5</int></value>
    </data></array></value></param>
    </params>
</methodCall>
```

Abbildung 3. Kodierung eines XML-RPC

Abbildung 3 zeigt als Beispiel, wie ein Aufruf mit einer 5-elementigen Integerliste kodiert wird. Deutlich ist der Aufwand zu erkennen, mit dem die zu übertragende Liste als XML-RPC Array kodiert wird. Die Dekodierung dieser Strukturen führt zu dem beobachteten Laufzeitverhalten mit den gravierenden Unterschieden zum reinen Funktionsaufruf ohne Datenübermittlung. Weitere Tests ergaben, dass die Rechenleistung, welche zum Kodieren der Requests benötigt wird wesentlich geringer ausfällt als die zur Dekodierung auf Serverseite benötigte Leistung.

Da Strukturen in XML-RPC als Array mit Parametern unterschiedlichen Typs kodiert werden, lassen sich die Aussagen direkt auf die Übertragung von Strukturen mittels XML-RPC abbilden.

2.3 Test 2: Transfer von Strings

Als letzter Test steht die Übertragung unstrukturierter Daten in Form von Strings auf dem Programm. Dazu wird eine entsprechende Methode des XML-RPC Servers mit Strings variabler Länge aufgerufen. Die Ergebnisse sind in Abbildung 4 dargestellt. Als Vergleich ist wieder die korrespondierende Realisierung mittels RMI dargestellt.

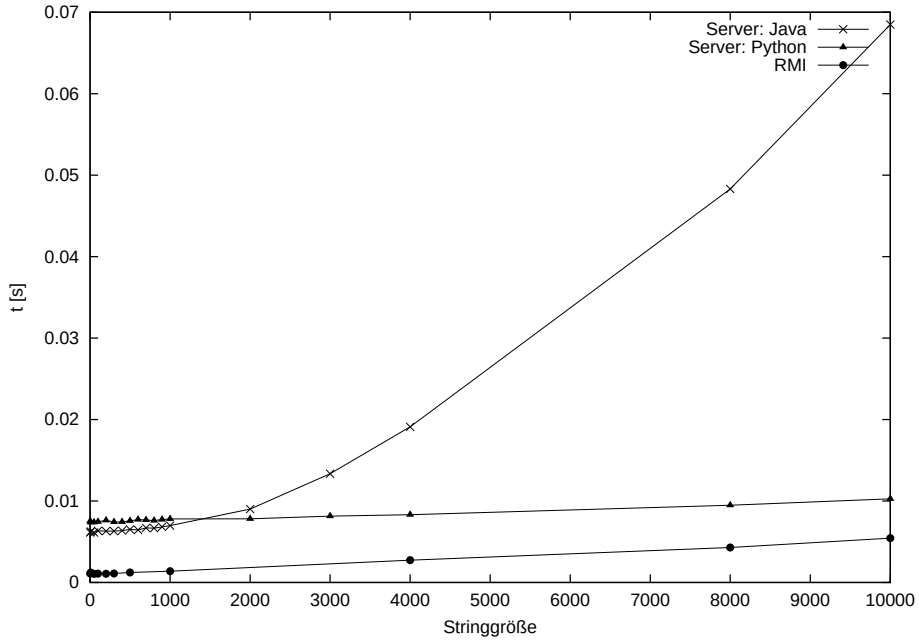


Abbildung 4. Ping mit Strings

2.3.1 Interpretation der Ergebnisse

Anhand der bisher betrachteten Ergebnisse überrascht die Tatsache, dass die Java-Implementierung trotz JIT um Größenklassen langsamer als die Pythonimplementierung ist. Desweiteren scheint der Anstieg nicht linear zu verlaufen. Das Ergebnis deutet auf ein Problem mit der gewählten Implementierung hin, zumal keine Strings mit mehr als 10000 Zeichen übertragen werden konnten. Weiterhin überrascht, dass die Pythonimplementierung im Vergleich zur RMI Realisierung fast vergleichbare Ergebnisse liefert und die Verarbeitungszeit mit zunehmender Stringgröße weit weniger steil ansteigt. Dies lässt den Schluss zu, dass XML-RPC für die Verarbeitung größerer Mengen unstrukturierter Daten weit besser geeignet ist als für die Verarbeitung der selben, als Array übertragenen Datenmenge.

3 Zusammenfassung

Die obigen Testfälle haben gezeigt, das die Kodierung und Übertragung von Remote Procedure Calls mittels XML einen nicht zu vernachlässigenden Anteil an Rechenleistung vor allem zur Dekodierung von Request benötigt. Die Tests erstrecken sich ausschliesslich auf die Betrachtung der Struktur der im RPC übergebenen Daten, lassen sich jedoch direkt auf Aussagen über das Verhalten analog strukturierter Responses zum Klient übetragen.

Allein der reine Methodenaufruf und die Übermittlung eines einzelnen Rückgabewertes erfordern einen teils um mehrere Größenklassen höheren Aufwand als vergleichbare andere Systeme. Dieser Mehraufwand liegt in der aufwendigen Dekodierung der XML Daten begründet. Die explizite Typkodierung der Übertragungsparemeter macht es notwendig, die Typen jeweils beim Empfang zu dekodieren. Durch die Typkodierung mittels XML sind dazu rechenzeitintensive Stringoperationen notwendig, um zunächst die XML-Tags zu lokalisieren, zu identifizieren und die kodierten Daten in eine programmierspracheninterne Repräsentation zu überführen.

XML-RPC eignet sich vor allem für die Übertragung von Daten, die keine starke interne Strukturierung aufweisen. Beim Transfer von längeren Listen oder umfangreichen Strukturen ist mit massiven Performanceeinbrüchen zu rechnen. Die Übertragung unstrukturierter Daten sollte weit geringeren Einfluss auf die benötigte Rechenleistung haben.

Interessant ist die Übertragbarkeit der getroffenen Aussagen auf alle nach ähnlichen Prinzipien wie XML-RPC arbeitenden Systemen. Hier ist vor allem das *Simple Object Access Protocol (SOAP)[3]* zu nennen, welches intern ähnliche Mechanismen wie XML-RPC verwendet, dabei aber wesentlich komplexer ist. Die zur Übertragungsleistung eines SOAP basierten Systems notwendigen Aussagen lassen sich so direkt aus den Aussagen über XML-RPC ableiten. Auch hier werden stark strukturierte, umfangreiche Daten zu ähnlichen Performanceanforderungen führen und einzelne Requests kaum schneller als in den gezeigten Beispielen abgearbeitet werden können.

Trotz allem bieten XML basierte RPC Mechanismen einen idealen Zugang zu wenig frequentierten Diensten oder zu Diensten, deren Aufruf und Rückgabeparameter einfach gehalten sind. Durch die Ansprechbarkeit aus den unterschiedlichsten Entwicklungssystemen lassen sich so schnell und unkompliziert verteilte Systeme in heterogenen Umgebungen realisieren.

Literatur

[1] *XML-RPC*. http://www.xmlrpc.org/.

[2] SCRIBNER, KENNARD: *Understanding SOAP*. Sams, 2000.

[3] *Simple Object Access Protocol (SOAP) 1.1*. http://www.w3.org/TR/SOAP/.

[4] FIELDING, R. und OTHERS: *RFC 2068: Hypertext Transfer Protocol – HTTP/1.1*, Januar 1997.

[5] PYTHONWARE: *XMLRPC Bibliothek für Python*. http://www.pythonware.org/.

[6] *Helma XML-RPC Library for Java.* http://xmlrpc.helma.org/.

[7] CORBIN, JOHN R.: *The Art of Distibuted Applications*. Springer Verlag, 1991.

[8] SRINIVASAN, R.: *RPC: Remote Procedure Call Protocoll Specification Version 2*, 1995. RFC 1831.