ISBN 978-89-6211-275-7

gSOAP 처리과정 및 성능 분석

(Analysis of gSOAP Processing Procedure and Performance)

금 복 희 (Bokhee Keum)
bhkeum@kisti.re.kr
Visualization Team, Supercomputing Center

한 국 과 학 기 술 정 보 연 구 원
Korea Institute of Science & Technology Information

제목 차례

표 차례

그림 차례

1. 서론

SOAP(Simple Object Access Protocol)[5]은 XML을 기반으로 하는 Web Service 용 통신 프로토콜로 프로그램 데이터를 XML로 표시하고 RPC(Remote Procedure Call, 원격 프로시저 호출)를 수행하는 방법을 정의하고 있다. 또한 SOAP 클라이언트에서는 호출 가능한 함수와 해당 함수에 전달되는 매개 변수가 포함되어 있으며 이런 정보는 XML로 변환이 되어 해당 서버에 전달이 된다. SOAP 서버는 실행된 함수의 결과와 함께 XML 메시지를 반환한다.

Simplicity, Robustness, Extensibility를 주 특징으로 하는 SOAP은 XML을 기반으로 하기 때문에 programming language와 platform에 독립적이다. 지리적으로 분산되어 있는 기관의 가상공간을 통한 연결, 컴퓨터 자원, 과학적 데이터의 공유가 Grid 기반의 기술과 Web 기술이 합쳐지면서 가능하게 되었다.

그러나 다루어야 할 데이터의 량이 방대한 Scientific Application에 SOAP을 적용하기에는 다음과 같은 문제점이 있다.

첫째, XML은 ASCII 형태로 모든 데이터가 표현되고 tag를 이용하여 각 element를 표현하므로 원래 전달하고자 하는 데이터의 량에 비해 변환된 메시지의 량이 상당히 커진다. 전송하고자 하는 데이터의 량이 크지 않을 경우에는 문제가 되지 않겠지만 simulation 데이터와 같이 용량이 수십 기가바이트(GB)에서 수십 테라바이트(TB)에 이르는 데이터의 경우에는 XML로 변환된 메시지의 크기로 인하여 전송 시간 지연이 훨씬 커진다.

둘째, 위와 같은 XML 메시지의 크기로 인하여 제어에 관련된 내용만 SOAP 메시지로 전달하고 전달할 데이터는 다른 프로토콜을 사용하는 대안을 사용하는 경우가 많다. 이와 같은 방법은 데이터 전송에 대한 문제는 해결되지만 서로 다른 모델을 함께 사용하기 때문에 개발하고 관리하는 데 대한 overhead가 발생하게 된다. 제어 관련 메시지의 전달은 SOAP를 사용하고 데이터는 GridFTP를 사용하는 경우를 예로 들 수 있다.

이와 같은 문제를 해결하기 위해서는 SOAP 메시지 처리에 대한 overhead와 메시지의 크기를 줄이는 방안이 필요하다. 따라서 이 글에서는 SOAP의 메시지 전달과정을 단계별로 나누어 살펴보고 여러 가지 형태의 데이터에 대해서 gSOAP[4]을 이용한 SOAP 메시지 처리 성능과 특징을 분석한 실험결과를 보여줄 것이다. 그리고 처리 단계 별로 적용할 수 있는 개선 방안을 마지막 절에서 소개한다. 이 실험결과를 바탕으로, 사용하고자 하는 응용분야에서 주로 다루는 데이터 형태를 중심으로 SOAP 성능을 개선할 수 있는 방안을 찾는데 도움이 되고자 한다.

2. SOAP 구조 분석

가. SOAP 메시지 처리 과정 분석

1) SOAP 메시지 전달 시

SOAP 클라이언트 메시지는 SOAP 서버로 전달되기 전 XML Serialization 과정을 통하여 XML 형식으로 변환된다. Serialization은 SOAP 클라이언트 프로그램에서 사용하고 있는 in-memory 오브젝트를 XML 스트림으로 변환하는 것을 말하며 이 스트림은 UTF-8/16과 같은 코드형태로 SOAP 서버에 전달된다. Serialization 과정을 다음과 같은 4단계로 나눌 수 있다[2].

단계 1. 전달할 메시지의 대상이 되는 데이터의 자료구조와 값을 찾음

단계 2. 사용자 시스템 형식에 맞게 표현되어 있는 데이터를 ASCII로 변환함

단계 3. ASCII로 변환된 문자열을 버퍼에 저장함

단계 4. 전송을 개시함

단계별로 자세한 내용을 살펴보면 다음과 같다.

단계 1 : Data Structure Traverse

SOAP 메시지로 전달되는 데이터는 사용자 프로그램에서 자료구조로 표현되고 각 각의 자료구조는 상응하는 SOAP XML 형식으로 변환된다. SOAP에서는 변환을 위하여 데이터의 자료구조와 값을 탐색한다.

단계 2 : Machine Representation to ASCII Conversion

SOAP 응용 프로그램에서 사용하는 데이터를 XML 메시지로 만들기 위해 ASCII로 변환한다. 예를 들어 “unsigned char chrbin[32]”와 “float fltbin[8]"는 HTTP 프로토콜로 전송할 때 다음과 같이 XML 메시지로 변환된다. 그림에서 배열의 element들은 하나씩 <item>, </item> 태그로 변환된 것을 볼 수 있다.

```
POST / HTTP/1.1
Host: localhost:50000
User-Agent: gSOAP/2.7
Content-Type: text/xml; charset=utf-8
Content-Length: 1025
Connection: close
SOAPAction: "SEND CHAR ARRAY"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:soapvdt"><SOAP-ENV:Body
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<ns:soapchar><chrbin xsi:type="SOAP-ENC:Array"
SOAP-ENC:arrayType="xsd:unsignedByte[32]"> <item>255</item>
<item>185</item><item>11</item><item>219</item><item>67</item><ite
m>133</item><item>22</item><item>167</item><item>201</item><item>
241</item><item>100</item><item>183</item><item>2</item><item>69</
item><item>42</item><item>175</item><item>121</item><item>235</ite
m><item>242</item><item>51</item><item>211</item><item>27</item>
<item>55</item><item>35</item><item>181</item><item>3</item><item
>5</item><item>114</item><item>160</item><item>235</item><item>17
7</item><item>159</item></chrbin></ns:soapchar></SOAP-ENV:Body></SO
AP-ENV:Envelope>
```

[그림 2] gSOAP의 char 형 배열에 대한 XML 메시지 변환 결과

```
POST / HTTP/1.1
Host: localhost:50000
User-Agent: gSOAP/2.7
Content-Type: text/xml; charset=utf-8
Content-Length: 725
Connection: close
SOAPAction: "SEND FLOAT ARRAY"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns="urn:soapvdt"><SOAP-ENV:Body
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<ns:soapfloat><fltbin xsi:type="SOAP-ENC:Array"
SOAP-ENC:arrayType="xsd:float[8]"> <item>32986356</item>
<item>-1.5231305E+09</item><item>60816744</item><item>3.21689498E+
09</item><item>3.12061645E+09</item><item>1.58120474E+09</item><ite
m>2.4064233E+09</item><item>1.60863885E+09</item></fltbin></ns:soapflo
at></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

[그림 3] gSOAP의 float 형 배열에 대한 XML 메시지 변환 결과

단계 3 : Writing ASCII to Buffer

SOAP에서 ASCII 형의 데이터를 buffer에 쓰는 과정은 다음과 같다.

```
sprintf(buf, "<integer>%d</integer>", value);
```

단계 4 : Sending SOAP Message

변환된 SOAP 메시지를 네트웍으로 전달하기 위하여 socket으로 Writing하는 과정이다.

gSOAP의 Serialization과정은 크게 SOAP 헤더 정보를 생성하는 SOAP header Serialization과 서버로 전달되는 매개변수를 SOAP 메시지로 변환하기 위한 SOAP body Serialization으로 나눌 수 있다. soapClient.cpp을 통해서 수행되는

XML 변환 과정을 주요 함수 별로 살펴보면 다음과 같다.

Function	Description
soap_serializeheader	SOAP의 header 정보 생성
soap_serialize body	SOAP의 body 정보 생성
soap_connect	SOAP의 HTTP header 등 HTTP 정보 생성
soap_envelope_begin_out	SOAP의 Envelope 시작 정보 삽입
soap_putheader	생성된 SOAP header 정보 삽입
soap_body_begin_out	SOAP의 body 시작 정보 삽입
soap_put_"service function name"	SOAP의 body 정보 삽입
soap_body_end_out	SOAP의 body 종료 정보 삽입
soap_envelope_end_out	SOAP의 Envelope 종료 정보 삽입

[표 1] SOAP Serialization에 사용되는 soapClient.cpp의 주요 함수

2) SOAP 메시지 수신 시

SOAP 응용 프로그램에서 전달되어 온 메시지를 처리하는 과정은 다음과 같이 나눌 수 있다.

단계 1. 네트웍에서 수신한 내용을 메모리 버퍼로 읽어 들임.

단계 2. XML 파싱과 해당 요소 처리

단계 3. ASCII 표현을 machine representation으로 변환.

단계 1 : Reading Message into Memory from OS Buffer

OS 버퍼로부터 SOAP 메시지를 읽어 들여 메모리 버퍼에 저장한다.

단계 2 : XML Parsing and Element Handling

버퍼로 읽어 들인 데이터는 XML 형식으로 되어 있으므로 XML tag와 value를 찾아서 관리해야 한다. XML을 처리하는 방법은 크게 두 가지가 있는데 하나는

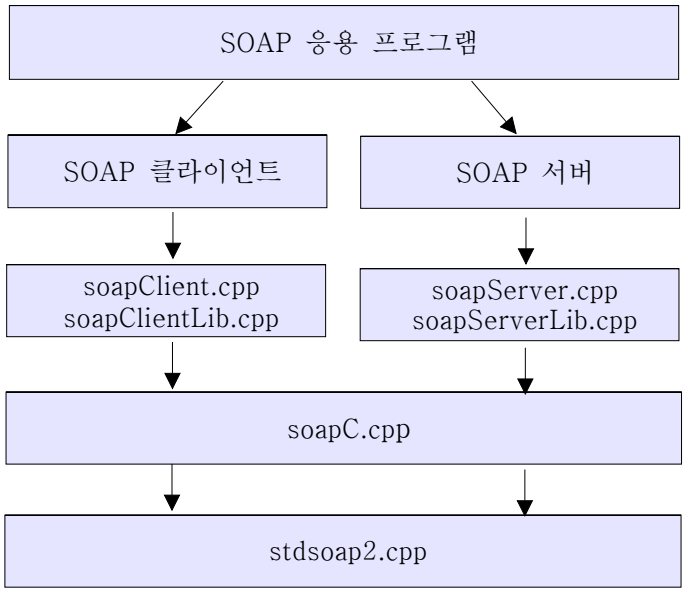
DOM(Document Object Model)이고 다른 하나는 SAX(Simple API for XML)이다. DOM은 메모리에 XML document의 object model을 만들어 모든 내용을 관리하는 방식이고 SAX는 callback을 통하여 element의 내용을 application에 알리는 방식이다. XML Pull Parser와 같은 Pulling Parsing이 있는데 이 방법은 완전한 object model을 만들지 않으면서(SAX처럼) tag와 content를 callback을 통하지 않고 직접 application에 전달한다.

단계 3 : ASCII to Machine Representation Conversion

ASCII representation 중에서 숫자 데이터는 decimal 표현에서 machine에 적합한 binary 표현으로 변환하고 문자 데이터는 UTF-8/16을 wide string으로 변환한다.

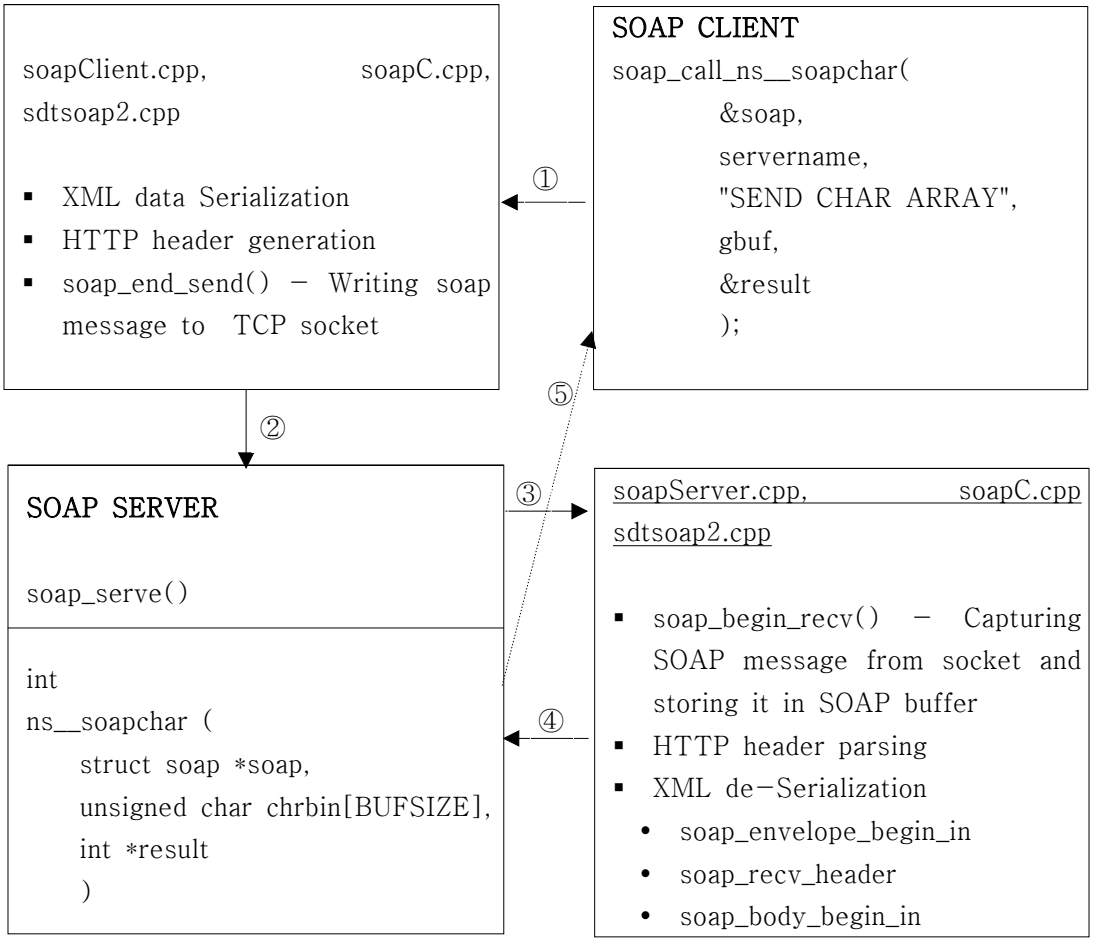
나. gSOAP 모듈의 계층 구조

gSOAP 응용 프로그램은 SOAP 서비스를 요청하는 클라이언트 프로그램과 서비스를 실행하여 결과를 반환하는 서버 프로그램으로 구성된다. SOAP 서비스를 정의하는 header 파일(예. soapexample.h)에 “soapcpp2”를 실행하면 (예. >soapcpp2 soapexample.h) [그림 3]에서와 같이 soapClient.cpp, soapClientLib.cpp, soapServer.cpp, soapServerLib.cpp, soapC.cpp 파일과 같은 SOAP stub 파일과 skeleton 파일이 만들어 진다. 그림에서 가장 아랫부분에 있는 stdsoap2.cpp 파일은 SOAP 응용 프로그램의 변화에 무관하게 독립적으로 유지되는 gSOAP 핵심 모듈이다.



[그림 4] gSOAP 모듈의 참조 구조

SOAP 클라이언트 프로그램에서 SOAP 서버에 있는 서비스, 예를 들어 ns__soapchar()를 요청하고 처리될 때까지 모듈 간 연결과정을 살펴보면 [그림 4]와 같다.



[그림 5] SOAP 서비스 호출과 처리과정

① SOAP 클라이언트 프로그램에서 SOAP 서버에 unsigned char array 형태의 gbuf를 전달하고 result의 결과 값을 기다린다.

② SOAP 서버 프로그램에서는 서비스 요청이 있는지 monitoring 하고 있다가 요청이 있을 때 soap_serve() 루틴을 실행한다.

③ 요청된 서비스를 실행 하기 위하여 SOAP 메시지를 처리하는 루틴을 호출한다.

④ 네트웍 버퍼로부터 메모리로 SOAP 메시지를 읽어 들이고 HTTP 프로토콜에 따라 content를 처리하고 XML deserialization을 수행한 다음 응용프로그램의 자료구조에 맞게 요소들을 변환하여 서비스 함수인 ns_soapchar()에 전달한다.

⑤ ns_soapchar() 실행이 끝나면 클라이언트에서 요청한 결과인 result를 전달한다.

3. SOAP 성능 분석

가. SOAP XML 메시지 크기의 Overhead

원 데이터(raw data)는 SOAP 서비스 모듈로 전달되기 전에 XML 형식의 메시지로 변환된다. 이때 변환된 메시지의 크기가 원 데이터에 비해 얼마나 증가하였는지를 알아보기 위해서 다음과 같이 세 가지의 실험을 실시하였다.

실험 1. 동일한 크기의 데이터에 대해서 서로 다른 데이터 유형의 배열을 전달함.

실험 2. 동일한 개수의 int 형 데이터를 각 각 배열과 binary 형으로 버퍼에 저장하여 전달함.

실험 3. 동일한 개수의 float, double 형 데이터에 대해서 배열에 저장하여 전달함.

SOAP 메시지 크기(XML의 크기, XMLsize)의 overhead는 원 데이터(raw data)에 대하여 변환된 메시지 크기의 증가율로 계산을 하였으며 계산식은 다음과 같다.

$$overhead(\%) = \left(\frac{XMLsize - RAWDATAsize}{RAWDATAsize} \right) \times 100$$

HTTP는 인터넷에서 가장 널리 사용되는 네트웍 프로토콜 중 하나로 반드시 요구되는 사항은 아니지만 SOAP의 하위 트랜스포트 프로토콜로 가장 많이 사용되고 있다. 이는 SOAP이 XML과 HTTP의 장점을 모두 가지면서 Grid Communication에 잘 활용될 수 있도록 한다. 따라서 여기서의 실험결과는 XML과 HTTP를 사용한 SOAP 메시지의 처리에 대해서 다룬다.

실험 장치

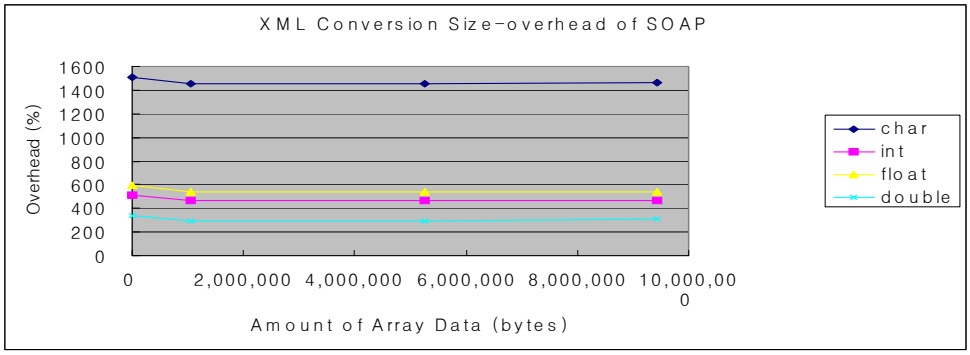
i686 processor x 2, Linux Federa Core6, 3.2GiB memory, 8.0GiB Swap, Intel 82545EM Gigabit Ethernet Controller

실험 1

1KiB, 1MiB, 5MiB, 9MiB 크기의 char, int, float, double 형 배열 데이터에 대해서 Serialization을 한 후 SOAP 메시지로 변환하였을 때 원 데이터(raw data)에 대해서 변환된 데이터의 크기와 overhead는 각각 [표 2], [그림 5]와 같다.

data type / data size	raw data	char	int	float	double
1KiB	1024	16489	6278	7085	4525
1024KiB	1048576	16327041	5893846	6749387	4092815
5120KiB	5242880	81634378	29468522	33744463	20461782
9216KiB	9437184	147622240	53043341	60734229	38666459

[표 2] 유형별 배열 데이터에 대한 XML 변환 메시지의 크기 (단위:byte)



[그림 6] 유형별 배열 데이터에 대한 XML 변환 메시지의 오버헤드(%)

double 형은 8바이트, int와 float 형은 4바이트, char 형은 1바이트이므로 동일한 크기의 데이터일 경우 데이터의 수는 double 형에 비해서 int와 float 형은 2배, char 형은 8배가 된다. 배열 데이터는 XML Serialization 시 <item>,</item> 태그로 하나의 element 씩 tag marking이 되므로 데이터의 개수가 가장 많은 char 형이 overhead가 가장 크고 데이터의 개수가 가장 작은 double 형이 overhead가 가장 낮다. 반면, int 형과 float 형은 데이터의 개수는 동일하지만 float의 경우 소수로 표현되므로 overhead가 int 형보다 더 크다.

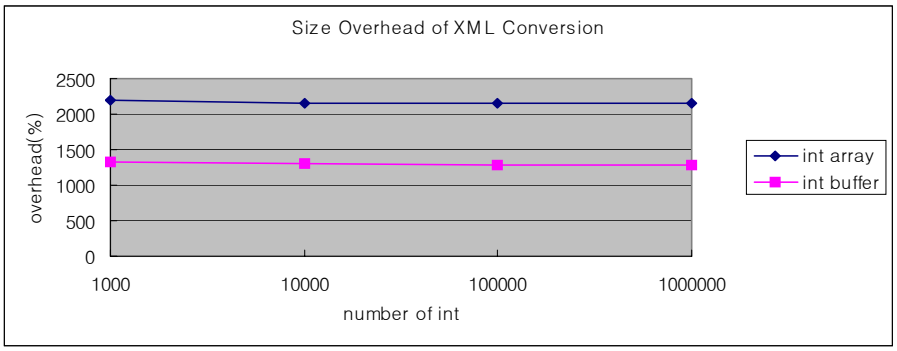
실험 2

int 형 데이터 1000, 10000, 100000, 1000000 개를 int 형 배열에 저장한 경우

("int-array")와 binary 형으로 변환하여 버퍼에 저장한 경우("int-buffer")에 대해서 XML Serialization을 수행했을 때 SOAP 메시지의 크기와 Overhead는 각각 [표 3], [그림 6]과 같다.

type / # of int	int-array(byte)	int-buffer(byte)	int-array to int-buffer ratio
1000	23023	14278	1.61
10000	225454	139563	1.62
100000	2250396	1392656	1.62
1000000	22499088	13927694	1.62

[표 3] int-array과 int-buffer의 SOAP 메시지 크기



[그림 7] int array, int buffer의 SOAP 메시지 Overhead

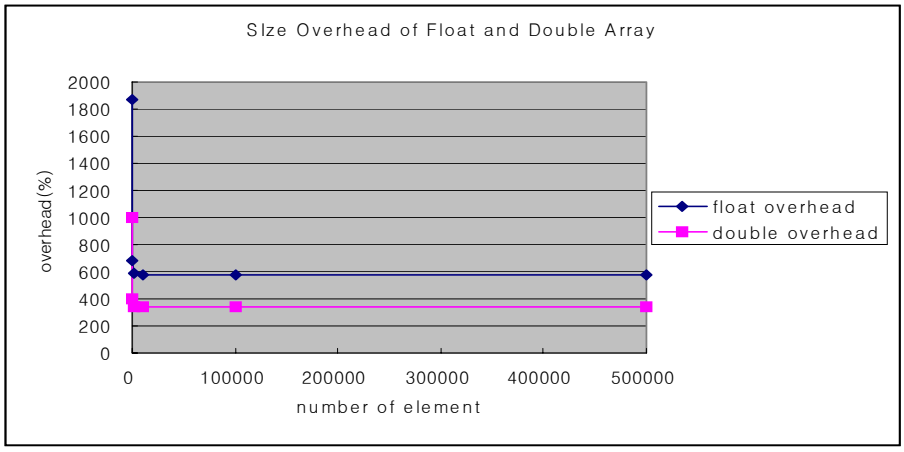
SOAP 메시지의 크기는 변환 전 원 데이터의 크기에 비해 int 형 배열로 전달될 경우는 약 2200 (%)로 증가하였고, binary 버퍼로 전달될 경우는 약 1300(%) 증가 되었다. overhead 값은 int형 데이터의 수에 상관없이 거의 일정함을 알 수 있다. SOAP 메시지로 변환하였을 때 int-array의 크기는 int-buffer의 크기에 대해 1.6배 더 크다.

실험 3

float 형과 double 형에 대해 각 각 10, 100, 1000, 10000, 100000 개의 element를 갖는 배열을 XML Serialization 했을 때 메시지의 크기와 overhead는 각각 [표 4], [그림 7]과 같다.

type # of element	float	double	ratio (double/float)
10	787	876	1.11
100	3142	4018	1.28
1000	27415	35419	1.29
10000	269391	349392	1.30
100000	2689454	3489254	1.30
500000	13445343	17444748	1.30

[표 4] float와 double의 SOAP 메시지의 크기와 double의 메시지 크기에 대한 float의 메시지의 크기 비율



[그림 8] float array, double array의 SOAP 메시지 Overhead

나. SOAP Serialization/Deserialization Performance

원 데이터(raw data)는 SOAP 서버에 전달되기 위해서 SOAP 메시지로 변환되어야 하며 수신한 SOAP 메시지를 SOAP 서비스 함수에서 이용하기 위해서는 원 데이터 형식으로 변환되어야 한다. 이와 같이 원 데이터를 XML 형식으로 변환되는 과정을 XML Serialization이라 하고 반대로 XML 형식의 메시지를 원 데이터로 변환하는 과정을 XML Deserialization이라 한다. 여기서는 XML Serialization과 Deserialization의 gSOAP 성능을 다음의 실험을 통하여 알아본다.

실험 1. 동일한 크기의 데이터에 대해서 서로 다른 네 가지 데이터 유형 즉, char, int, float, double 형에 대해서 Reading/Writing Time, De-/serialization 성능 측정.

실험 2. 동일한 개수의 int 형 데이터에 대해서 배열과 binary 버퍼의 De-/serialization 성능 측정.

실험 3. 동일한 개수의 float, double 형 데이터에 대한 De-/serialization 성능 측정.

실험 장치

i686 processor x 2, Linux Federa Core6, 3.2GiB memory, 8.0GiB Swap, Intel 82545EM Gigabit Ethernet Controller

실험 1

SOAP Writing Time

동일한 크기의 char, int, float, double 형 데이터를 SOAP 메시지로 변환한 다음 TCP socket에 Writing 하는데 걸리는 시간을 알아보자. Writing Time은 SOAP 서비스를 호출한 시점으로부터 soap_end_send() 수행이 끝날 때의 시간으로 하였다. 이 때 SOAP 메시지의 크기가 65536 바이트 이상일 경우에는 TCP/UDP

payload 크기 때문에 여러 번 나누어 socket writing이 이루어지므로, writing time은 마지막 socket writing이 끝날 때까지 걸리는 시간이 된다.

SOAP Reading Time

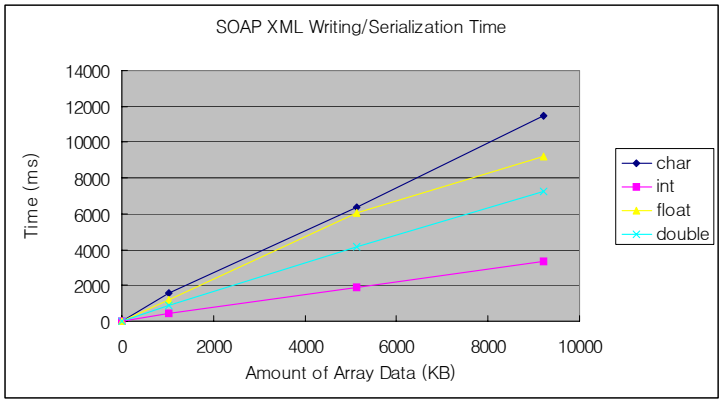
SOAP 서비스 모듈이 네트웍을 통해 전달된 데이터를 TCP socket으로부터 읽어들여서 XML 형식의 SOAP 메시지를 응용프로그램의 원 데이터(raw data)로 변환하는데 소요되는 시간을 측정하였다.

size / type	1KiB	1024KiB	5120KiB	9216KiB
char	3.00	1603.30	6343.30	11468.42
int	0.98	410.31	1868.35	3361.42
float	3.32	1221.51	6050.75	9178.69
double	2.84	906.14	4155.38	7267.68

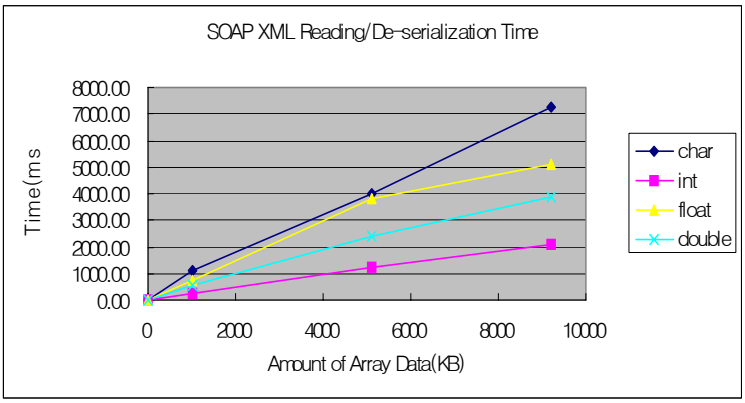
[표 5] SOAP Writing Time [ms]

size / type	1KiB	1024KiB	5120KiB	9216KiB
char	1.63	1115.67	4019.68	7287.40
int	0.56	256.14	1210.73	2112.59
float	2.54	755.57	3804.33	5093.44
double	0.71	549.63	2377.79	3875.57

[표 6] SOAP Reading Time [ms]



[그림 9] SOAP 메시지 Writing Time



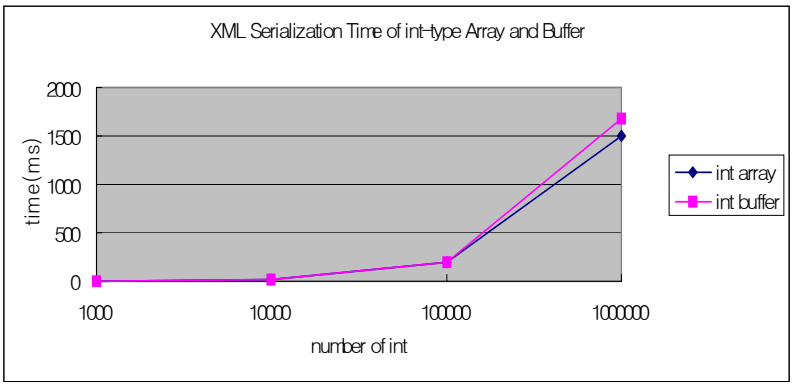
[그림 10] SOAP 메시지 Reading Time

실험 2

동일한 수의 int 데이터에 대해서 int-array와 int-buffer에 각각 저장하여 SOAP 메시지로 전송할 때 소요되는 XML Serialization/Deserialization Time과 Writing/Reading Time, 그리고 Writing/Reading에 소요되는 시간 중 Serialization/Deserialization Time이 차지하는 비율에 대해서 조사하였다.

type / # of int	int-array	int-buffer	diff(arr-buf)
1000	2186	2533	-347
10000	15200	16966	-1766
100000	196753	202073	-5320
1000000	1497750	1680737	-182987

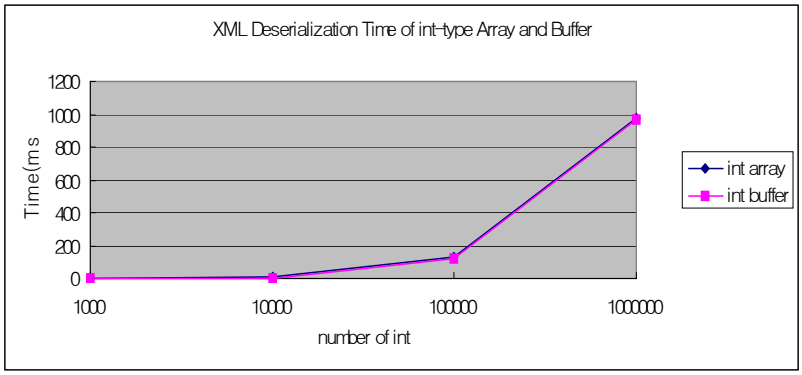
[표 7] int-array와 int-buffer의 XML Serialization Time [us]. 마지막 열의 diff(arr-buf)는 int-array 처리시간에서 int-buffer 처리시간을 뺀 값임



[그림 11] int array와 binary buffer의 XML Serialization Time

type / # of int	int-array	int-buffer	diff(arr-buf)
1000	780	347	433
10000	7851	3229	4622
100000	130142	121132	9010
1000000	978458	967279	11179

[표 8] int-array와 int-buffer의 XML Deserialization time. 마지막 열의 diff(arr-buf)는 int-array 처리시간에서 int-buffer 처리시간을 뺀 값임. [us]



[그림 12] int array와 binary buffer의 XML Deserialization Time

XML Serialization의 경우 int-buffer 데이터에 대한 처리 시간이 int-array 데이터를 처리하는 시간보다 더 오래 걸리며 데이터의 개수가 증가함에 따라 Serialization Time이 선형적으로 증가함을 알 수 있다. Deserialization의 경우, int-buffer 데이터를 처리하는 시간이 int-array의 경우보다 더 적게 걸리며

Serialization의 경우와 마찬가지로 데이터의 개수가 증가함에 따라 소요되는 시간이 선형적으로 증가한다. 또한 Serialization 처리 시간이 Deserialization 처리 시간보다 더 오래 걸린다.

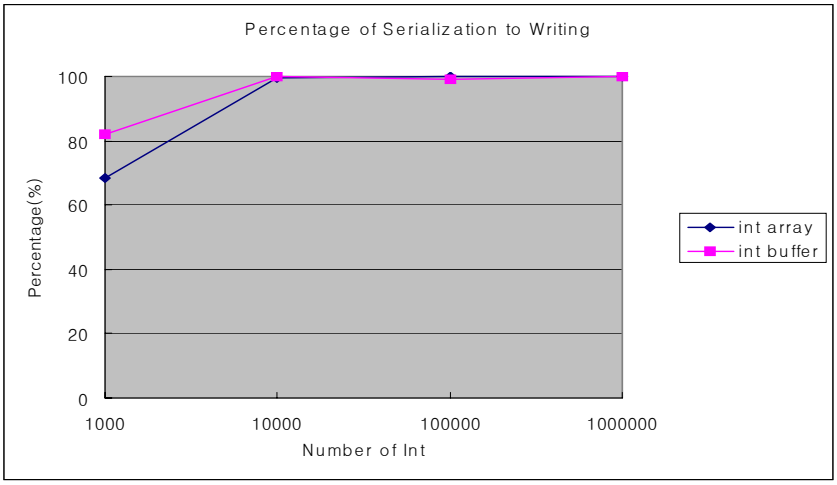
type / # of int	int-array	int-buffer
1000	2.80	7.30
10000	1.94	5.25
100000	1.51	1.67
1000000	1.53	1.74

[표 9] XML Deserialization 처리 시간에 대한 Serialization 처리 시간의 비율($\frac{Serialization\ Time}{Deserialization\ THme}$)

Serialization과 Deserialization의 처리 시간이 각각 SOAP 메시지 Writing과 Reading 처리 시간에서 차지하는 비율은 데이터의 크기가 클수록 커지며 전체 소요시간의 99% 이상을 차지하고 있다.

type / # of int	int-array	int-buffer
1000	68.35522	81.8946
10000	99.77027	99.9234
100000	99.98221	99.1531
1000000	99.99773	99.9971

[표 10] SOAP 데이터 Writing에 걸리는 시간 중 Serialization 시간이 차지하는 비율

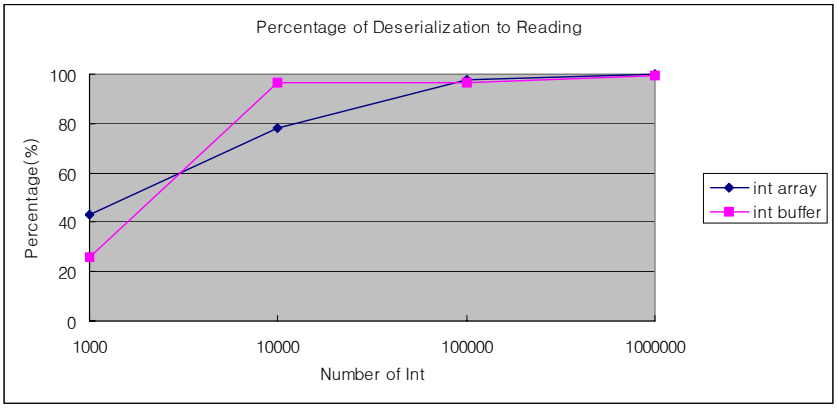


[그림 13] SOAP 메시지 Writing에서 Serialization이 차지하는 비율

int 형의 경우 배열로 저장하여 보내는 것이 binary로 저장하여 보내는 것 보다 XML Serialization Time은 적게 걸리고, XML Deserialization Time은 더 많이 걸린다. Deserialization Time은 Serialization Time에 비해 훨씬 적은 시간이 소요되고 배열로 저장되어 있는 int 형 데이터를 binary로 변환하여 보낼 경우 전송해야 할 SOAP 메시지의 크기는 int-array 보다 60% 더 작아지지만([표 3]) Serialization/Deserialization 처리 시간을 줄이는데 크게 영향을 주지 않는 것을 알 수 있다. 따라서 int형 데이터는 binary 형으로 변환하지 않고 그냥 전송하는 이 Serialization/Deserialization time 측면에서는 효율적이다.

type / # of int	int-array	int-buffer
1000	43.07013	25.95363
10000	77.92556	96.50329
100000	97.86069	96.79332
1000000	99.71902	99.57884

[표 11] SOAP 데이터 Reading에 걸리는 시간 중 Deserialization 시간이 차지하는 비율



[그림 14] SOAP 메시지 Reading에서 Deserialization이 차지하는 비율

그러나 클라이언트/서버간 SOAP 메시지의 송수신 성능은 XML Serialization/Deserialization 과정뿐만 아니라 네트웍을 통한 송수신도 고려가 되어야 하므로 전체적인 SOAP 메시지의 송수신 성능 평가를 위해서 다음과 같은 시스템 구성 하에 동일한 수의 정수형 데이터에 대해서 int-array, int-buffer 형으로 SOAP XML 변환을 통한 메시지 전송 시간을 측정하였다.

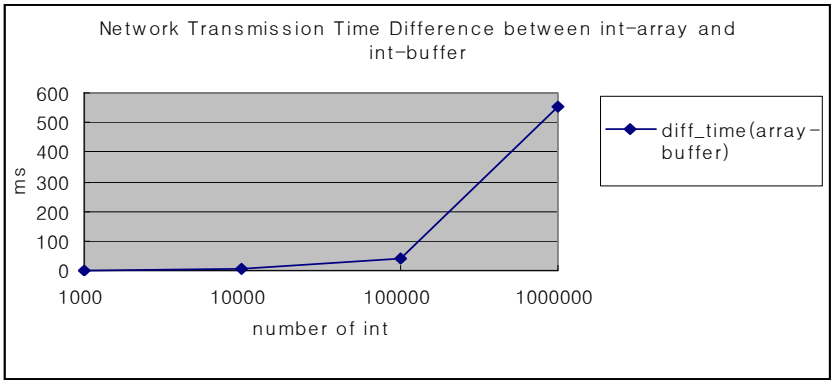
end-to-end SOAP 메시지 전송 비용(cost) = serailziation cost + communication cost over the network + Deserialization cost

SOAP 클라이언트 node(stereo) - i686 processor x 2, Linux Federa Core6, 3.2GiB memory, 8.0GiB Swap, Intel 82545EM Gigabit Ethernet Controller

SOAP 서버 node - octa cluster

# of int	diff(array-buff) [ms]
1000	0.726
10000	3.036
100000	42.223
1000000	556.253

[표 12] int-array, int-buffer 데이터의 SOAP 메시지 네트웍 전송 시간차[ms]



[그림 15] int-array와 int-buffer의 SOAP메시지의 네트웍 전송시간 차 : 시간차 = int_array time - int_buffer time

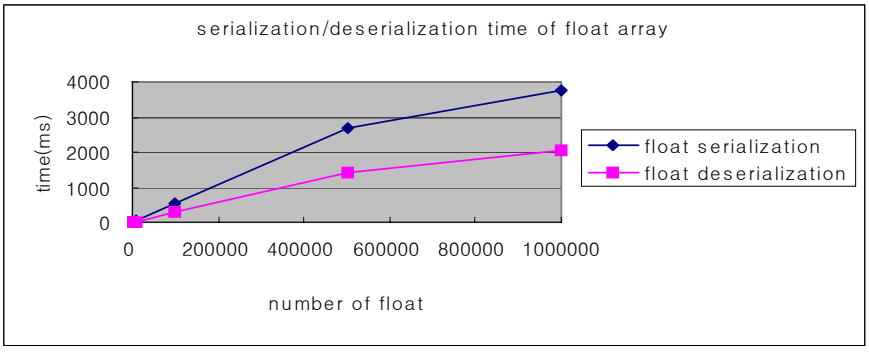
[표 12]와 [그림 14]에서 알 수 있듯이 int-array형 데이터의 전송 시간이 int-buffer형 데이터의 전송시간 보다 길며 데이터의 개수가 증가함에 따라 선형적으로 증가함을 알 수 있다. 10^9개의 int 형 데이터의 전송시간은 int-array가 int-buffer보다 약 525초 정도 더 소요되었으며 이 값은 10^6개 데이터의 경우보다 10^3배 더 소요된 시간이다.

	time [sec]
int-array	3040.764
int-buffer	2515.556
diff(array-buffer)	525.207

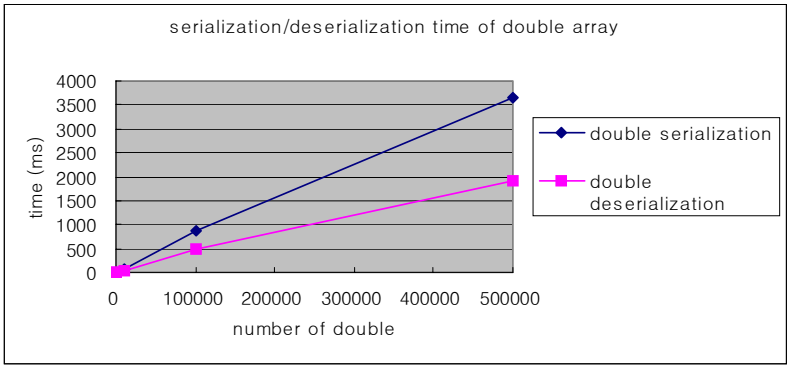
[표 13] 10^9개의 int 형 데이터 전송 시간의 비교

실험 3

floating point 수의 De-/Serialization 처리 시간을 알아보기 위하여 float 형과 double 형으로 나누어 각각 동일한 수의 데이터에 대하여 배열을 생성한 후 De-/Serialization 시간을 알아보았다.



[그림 16] float 형 배열의 De-/Serialization time



[그림 17] double 형 배열의 De-/Serialization time

type / # of element	float	double
10	6.21	3.56
100	5.60	5.98
1000	5.00	6.80
10000	2.61	2.29
100000	1.92	1.74
500000	1.90	1.91

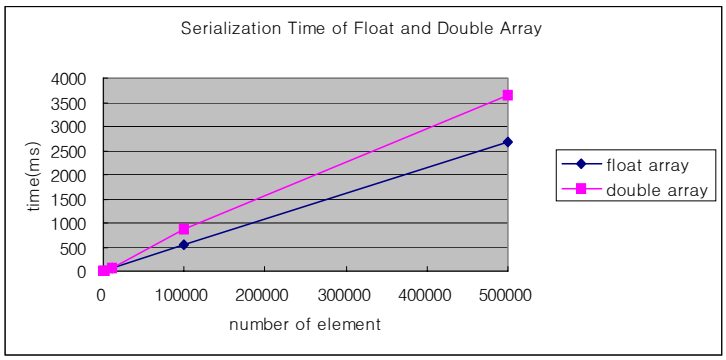
[표 14] float, double 형 array의 Serialization과 Deserialization의 처리시간 비율 : $\frac{serialization\,time}{deserialization\,time}$

1. float 형 배열의 경우 Serialization time은 Deserialization time보다 더 많이 소요된다[그림 15].

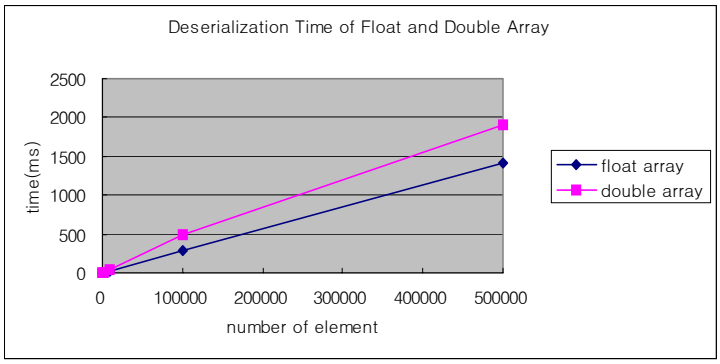
2. double 형 배열의 경우도 마찬가지로 Serialization time이 Deserialization time보다 더 많이 소요된다[그림 16].

3. float 형의 Serialization time(Deserialization time)과 double 형의 Serialization time(Deserialization time)의 경우 double 형이 더 많이 소요된다[그림 17, 18].

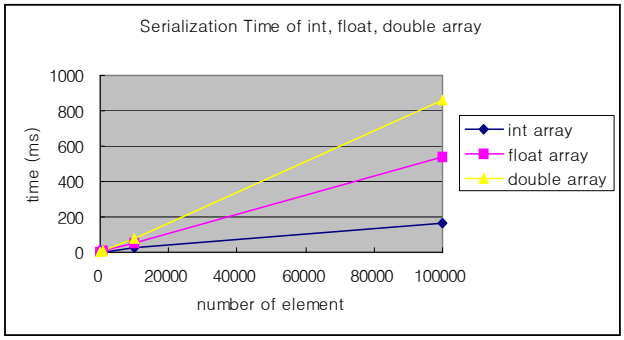
4. int 형과 floating point 형의 처리시간의 경우 element의 수가 10개인 경우를 제외하고 element의 개수가 증가함에 따라 de-/serialization 처리 시간 비율이 점진적으로 증가하고 이 현상은 Deserialization에서 보다 Serialization에서 두드러짐을 알 수 있다.



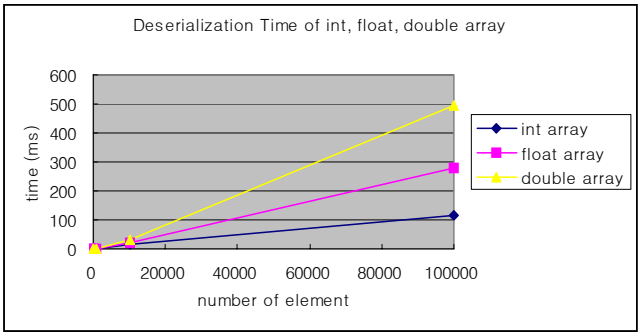
[그림 18] float 형 배열과 double 형 배열의 Serialization Time 비교



[그림 19] float 형 배열과 double 형 배열의 Deserialization Time 비교



[그림 20] int, float, double 형 배열의 Serialization Time 비교



[그림 21] int, float, double 형 배열의 Deserialization Time 비교

type / # of element	float/int	double/int
10	0.91	0.21
100	1.75	1.17
1000	2.83	4.25
10000	2.49	3.49
100000	3.16	5.08

[표 15] int array의 Serialization time에 대한 float, double array의 Serialization time의 비

type / # of element	float/int	double/int
10	1.21	0.48
100	1.77	1.11
1000	1.63	1.81
10000	1.54	2.45
100000	2.42	4.30

[표 16] int array의 Deserialization time에 대한 float, double array의 Deserialization time의 비

4. SOAP 메시지 처리 성능 개선 방안

가. SOAP 메시지 전송 시

1) Floating Point의 Serialization Time 개선

데이터의 ASCII변환에서 가장 많은 시간이 소요되는 것은 IEEE 754 floating point 데이터를 ASCII로 변환하는 것이다. 이 부분은 전체 end-to-end communication 시간에서 90%를 차치한다[2]. floating point의 Serialization 시간을 줄이기 위해 주로 제시되는 방법은 ASCII representation 대신 binary representation[1, 3]을 지원하는 것이다.

2) ASCII to Buffer Writing 개선

SOAP에서 데이터를 버퍼에 writing하는 방식은 다음과 같다.

```
sprintf(buf, "<integer>%d</integer>", value);
```

이와 같이 할 경우, buf에 저장될 값들은 먼저 메모리에 저장된 후 그 다음에 buf에 저장이 된다. 따라서 이를 개선하기 위해서는 다음과 같이 코드를 작성하는 것이 좋다.

```
*buf++ = '<';
*buf++ = 'i';
*buf++ = 'n';
    . . .
```

3) SOAP 메시지 Sending 개선

HTTP 1.0의 경우 HTTP body의 크기를 Content-Length라는 header field에 명시를 해야 하기 때문에 XML Serialization이 모두 끝나야 Content-Length의 값을 설정할 수 있다. SOAP 메시지의 전달과정은 일반적으로 두 단계로 나눌 수 있다. 첫째, 두 개의 버퍼를 설정하여 하나는 HTTP header 정보를 저장하고

다른 하나는 HTTP body인 XML 정보를 저장한다. 둘째, Content-Length field의 값을 알아내기 위해 XML Serialization을 먼저 수행한 다음 HTTP header 정보를 완성한다. 이 경우 두 번의 system call이 요구될 뿐만 아니라 메시지가 클 경우 요구되는 메모리가 많아진다. 또한 TCP/IP의 경우 데이터 전송을 위해서는 먼저 TCP connection이 형성되어야 하며 이를 위해 최소 하나의 패킷(예. SYN)을 교환하는 과정이 필요한데, system call을 여러 번 할 경우 서로 다른 connection을 형성한다면 메시지 전달에 있어서 round-trip delay가 생길 수 있다. 게다가 OS resource를 소모할 수 있다. TCP/IP의 경우 closed connection시 한쪽 node의 상태가 TIME_WAIT의 상태가 maximum segment lifetime의 두 배에 해당하는 시간동안 유지되며 이 시간은 길게는 4분 정도 소요된다. 이 기간 동안 일부 메모리가 유지되어야 하고 사용된 포트도 바로 이용할 수 없는 상태가 된다.

system call을 줄이기 위해서는 하나의 버퍼를 사용할 수 있어야 하는데, 이는 back-patching 혹은 vectored sends를 쓰면 해결될 수 있다.

back-patching : 버퍼에 Content-Length 값을 쓰는 부분을 공백으로 비워두고 이 후 XML Serialization이 끝나면 그 부분을 채워 놓는 것.

vectored sends : UNIX와 Win32 machine에서 지원되는 것으로 한번의 send call에 서로 다른 메모리 버퍼를 결합해서 보낼 수 있는 방법.

두 번째 문제는 HTTP 1.1을 사용하거나 two-step Serialization으로 해결될 수 있다.

HTTP 1.1 : HTTP 1.1은 chunked encoding을 지원하는데 이는 HTTP body부분을 각 chunk의 크기와 함께 여러 개의 chunk로 보내는 것으로 Content-Length가 필요하지 않다. HTTP 1.1을 사용할 경우 한번에 body내용을 모두 버퍼에 저장하지 않아도 된다. 대부분의 운영체제는 send system call에서 사용자 메모리 버퍼의 내용을 Kernel 영역의 버퍼에 쓴 다음 바로 return하고 실제 전송작업은 background에서 이루어진다. 대량의 메시지 전송 시 데이터 전송과 Serialization을 병행할 수 있다는 장점이 있으나 여러 번의 system call로 인

하여 overhead가 발생할 수 있다.

Two-Step Serialization : 첫 번째 단계에서는 메모리에 실제로 Serialization 내용을 저장하지 않고 Serialization의 길이만 구하여 HTTP header를 완성하여 버퍼에 저장하고 그 다음 Serialization 내용을 저장하는 것. HTTP body는 전체내용이 버퍼에 모두 한 번에 있을 필요가 없으므로 필요한 메모리를 줄일 수 있고 하나의 버퍼를 사용하므로 system call을 줄일 수 있음.

나. SOAP 메시지 수신 시

1) OS 버퍼에서 SOAP 버퍼로의 Writing 개선

OS 버퍼로부터 SOAP 메시지를 읽어 들여 메모리 버퍼에 저장한다. 이 과정에서 system call을 줄이기 위해서는 한 번에 많은 데이터를 읽어 들이는 것이 좋다.

2) XML Parsing과 Element Handling 개선

XML Parser의 성능을 개선하기 위해 [2]에서는 다음과 같은 방법을 제안하고 있다.

(1) tag 탐색에 Trie이용 : STL(Standard Template Library)을 사용한 경우와 Trie의 성능을 비교해 놓았는데 int와 double 데이터가 같이 있는 경우, Linux 시스템에서는 STL에 비해 31.5%, Solaris에서는 22.7% 성능이 향상 되었다. double은 없고 int만 있는 데이터의 경우 Linux에서는 51.2%, Solaris에서는 50.8%로 성능이 향상되었으며 두 시스템에서의 성능개선이 비슷하게 이루어졌다. Tag는 전체 SOAP 메시지에서 많은 부분을 차지하므로 Trie를 이용한 tag parsing은 Deserialization에 크게 도움이 된다.

(2) Schema에 따른 Parsing : 예를 들어, 배열을 위한 별도의 Array Parser를 구현한다. Array Parsing이 지원될 경우 Linux와 Solaris에서 모두 성능이 개선되었으며 double 데이터 처리 때문에 Linux에 비해 Solaris의 성능 개선 폭은 작은 편이다.

References

[1] Kenneth Chiu, Tharaka Devadithya, Wei Lu, Aleksande Slominski, "A Binary XML for Scientific Applications", e-Science 2005.

[2] K. Chiu, M. Govindaraju, and R. Bramley. "Investigating the Limits of SOAP Performance for Scientific Computing", In Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing.

[3] Wei Lu, Kenneth Chiu, Dennis Gannon, "Building a Generic SOAP Framework over Binary XML", IEEE HPDC 2006.

[4] gSOAP, http://www.cs.fsu.edu/~engelen/soap.html

[5] D. Box et al. Simple object access protocol 1.1, 2000. http://www.w3.org/TR/SOAP.