Monitoring Distributed Traces with Kieker

Hannes Strubel

June 1, 2017
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
Motivation

Client

Request

Response

Server
Goals

Motivation

- Design a concept for monitoring distributed traces
- Implement the monitoring
- Implement the trace reconstruction
- Evaluate the concept
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
- Kieker Monitoring Framework
- AspectJ
- TeeTime
BeforeOperationEvent main()
BeforeOperationEvent StringBuilder.append(..)
AfterOperationEvent StringBuilder.append(..)
BeforeOperationEvent StringBuilder.toString()
AfterOperationEvent StringBuilder.toString()
BeforeOperationEvent PrintStream.println(..)
AfterOperationEvent PrintStream.println(..)
AfterOperationEvent main()
Communication Technologies

Foundations

- REST - Jax-RS
- SOAP - Jax-WS
- RPC - RMI
- Message-oriented middleware - Apache ActiveMQ
Approaches for Tracing

Foundations

- Metadata propagation
- Schema-based
- Black-box inference

RR Sambasivan, R Fonseca, I Shafer, GR Ganger - So, you want to trace your distributed system? Key design insights from years of practical experience 2014
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
Monitoring RPC

Related Work

- Exchange trace ID
- Generate records for sending and receiving a message
- Reconstruct traces with the additional records

SOAP Probe

Related Work

- Transfer information (e.g. trace ID, order ID)
- Modify SOAP header
- Continue client trace on server-side
- Use Apace CXF Interceptor
Tracing Tools

Related Work

Google’s Dapper
A Large Scale Distributed Systems Tracing Infrastructure

X-Trace
Network diagnostic tool

MagPie
Modelling and performance-aware systems
Twitter’s OpenZipkin

based on Google’s Dapper paper¹

<table>
<thead>
<tr>
<th>Language</th>
<th>Library</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td>zipkin-go-opentracing</td>
<td>Go kit, or roll your own with OpenTracing</td>
</tr>
<tr>
<td>Java</td>
<td>brave</td>
<td>Jersey, RestEASY, JAXRS2, Apache HttpClient, Mysql</td>
</tr>
<tr>
<td>JavaScript</td>
<td>zipkin-js</td>
<td>cujoJS, express, restify</td>
</tr>
<tr>
<td>Ruby</td>
<td>zipkin-ruby</td>
<td>Rack</td>
</tr>
<tr>
<td>Scala</td>
<td>zipkin-finagle</td>
<td>Finagle</td>
</tr>
</tbody>
</table>

¹BH Sigelman, LA Barroso, M Burrows, P Stephenson - Dapper, a Large-Scale Distributed Systems Tracing Infrastructure 2010
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
Monitor TCP events

Approach

- Communication Start
  - Client: connect()
  - Server: accept()

- Outgoing Communication
  - Client: write()
  - Server: read()

- Incoming Communication
  - Client: read()
  - Server: write()

- Communication End
  - Client: close()
  - Server: close()
Aggregate TCP events

Approach

Client

Operation

Communication Start

Outgoing Communication

Incoming Communication

Communication End

Operation

Server

Communication Start

Incoming Communication

Server-side Operations

Outgoing Communication

Communication End

Operation
Merge Traces

Approach

Client

Client-side Operation

Server

Server-side Operations

Client (merged)

Client-side Operation

Client-side Operation

Server-side Operations

Client-side Operation
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
Goals of the evaluation

- Monitor and reconstruct TCP communication
- Monitor REST, SOAP, RMI and reconstruct distributed traces
Pointcuts for a TCP communication

Evaluation

- Socket.connect()
- Socket.getOutputStream()
- DataOutputStream.new(OutputStream)
- Socket.getInputStream()
- InputStreamReader.new(InputStream)
- BufferedReader.new(InputStreamReader)
- DataOutputStream.write()
- BufferedReader.read()
- Socket.close()
TCP reduced call-tree

Evaluation

TCP-Server::
@0:TCP-Server
public static void execute()

  TCP-Server::
@1:ServerSocket
public Socket accept()

  TCP-Server::
@1:Socket
public OutputStream getOutputStream

  TCP-Server::
@1:Socket
public OutputStream getInputStream

  TCP-Server::
@1:BufferedReader
public final String readLine()

  TCP-Server::
@1:DataOutputStream
public final void writeBytes(String)

TCP-Client::
@0:TCP-Client
public static void execute()

  TCP-Client::
@1:Socket
public void connect(SocketAddress)

  TCP-Client::
@1:Socket
public OutputStream getOutputStream

  TCP-Client::
@1:Socket
public OutputStream getInputStream

  TCP-Client::
@1:DataOutputStream
public final void writeBytes(String)

  TCP-Client::
@1:BufferedReader
public final String readLine()

TCP-Server::
@1:Socket
public synchronized void close()
GET method from a restful service

```java
@GET
@Produces(MediaType.TEXT_PLAIN)
public String halloPlainText()
{
    return "Hello World";
}
```
Reduced call-tree

Evaluation

Entity

TCP-Client::
@0:javarsexample.RestfulClient
public void sendGetToService()

TCP-Client::
@1:HttpURLConnection
public int getResponseCode()

TCP-Server::
@2:StringBuilder
public StringBuilder append()

TCP-Server::
@2:StringBuilder
public StringBuilder append()

TCP-Server::
@2:StringBuilder
public String toString()
WebMethod from a SOAP webservice

1  @WebMethod
2  public String
    getHelloAsString() {
3      return "Hello World";
4  }

Interface for an example RMI service

```java
public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
```

Implementation for an example RMI service

```java
public class Server implements Hello {
    public String sayHello() throws RemoteException {
        return "Hello world!";
    }
}
```
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work
TCP probes and reconstruction of asynchronous trace works

Probes do not apply for other communications

therefore, probes for every implementation required

AspectJ is maybe not adequate

IP address is not a proper identifier in the application-layer
1. Motivation

2. Foundations

3. Related Work

4. Approach

5. Evaluation

6. Conclusion

7. Future Work