Cloud Services

Roy Campbell

Acknowledgements to: Shivaram Venkataraman, Emilio Zegarra, Barone Budge and Dominique, Mirko Montanari, Curt Kersey, Aron Silverton

1/25/2013
Introduction to Web Servers & Load Balancing

- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server
Contents

• Remote Procedure Calls
  – RMI, SOAP
• HTTP, REST
• JSON, XML
• Load Balancing Architectures
• Load Balancing Schedulers
Middleware Layers

- Applications
- RPCs and RMIs, e.g., CORBA; SOAP
- Request reply protocol
- External data representation
- Operating System

Middleware layers

Provide support to the application
Run at all servers @user level

RMI=Remote Method Invocation
CORBA=Common Object Request Brokerage Architecture
SOAP=Simple Object Access Protocol
Middleware Layer Definition

• **Software that provides services to applications beyond those generally available at the operating system**

• Middleware implement functionalities that are common across many different application
  – No need of reinventing the wheel (e.g., message parsing!) every time that you need to do something

• Can we build distributed systems while maintaining our code not very different from a single-node program?
Common Abstractions

• In single-node programs, we often rely on concepts such as:
  – Procedure calls
  – Objects
  – Shared memory
  – ...

• A Middleware Layer can provide the same abstractions to distributed applications!

• We’ll start with objects and procedure calls
Local Objects

• Within one process’ address space
• Object
  – consists of a set of data and a set of methods.
  – E.g., C++ object, Chord object at a client (Chord data structures + functions at a node).
• Object reference
  – an identifier via which objects can be accessed.
  – i.e., a pointer
• Interface
  – provides a definition of the signatures of a set of methods (i.e., the types of their arguments, return values, and exceptions) without specifying their implementation.
  – E.g., \{ put(objectname), get(objectname)\} interface for Chord object. Same interface also applies to other P2P objects such as Gnutella, Kazaa, etc.
Remote Objects

• May cross multiple process’ address spaces
• Remote method invocation
  – method invocations between objects in different processes (processes may be on the same or different host).
  – Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system
• Remote objects
  – objects that can receive remote invocations.
• Remote object reference
  – an identifier that can be used globally throughout a distributed system to refer to a particular unique remote object.
• Remote interface
  – Every remote object has a remote interface that specifies which of its methods can be invoked remotely. E.g., CORBA interface definition language (IDL).
Request-reply communication

Client

- doOperation
- (wait)
- (continuation)

Server

- getRequest
- select object
- execute method
- sendReply

Request
- message

Reply
- message
A Remote Object and Its Remote Interface

Example Remote Object reference=(IP,port,objectnumber,signature,time)
Remote and Local Method Invocations

**Local invocation** = between objects on same process.
Has *exactly once* semantics

**Remote invocation** = between objects on different processes.
Ideally also want *exactly once* semantics for remote invocations
But difficult in distributed systems (why?)
Failure Modes of RMI RPC

Correct function:
- Execute
- Reply

Lost request:
- Request

Crash before reply:
- Execute
- Crash
- Reply

Crash before execution:
- Request
- Crash
- Reply

Channel fails during reply:
- Execute
- Reply

Client machine fails before receiving reply:
- Execute
- Reply

(and if request is received more than once?)
Invocation Semantics

- Middleware aims at providing **transparency**
  - Sometimes obtaining transparency is expensive and unnecessary
  - E.g., because of faults, obtaining “exactly-once” semantic is expensive

- Other invocation semantics are possible:
  - **Maybe**: caller can not determine whether or not the remote method has been executed
  - **At-least-once**: caller either receives a result (in which case the user knows the method was executed at least once) or an exception
  - **At-most-once**: caller either receives a result (in which case the user knows the method was executed at most once) or an exception

- The right invocation semantic depends on the application
  - For example, at-least-once semantic + **idempotent operations** can provide a semantic that would be equivalent to exactly-once to applications

Idempotent=same result if applied repeatedly, w/o side effects
# Invocation Semantics

<table>
<thead>
<tr>
<th>Retransmit request message</th>
<th>Duplicate filtering</th>
<th>Re-execute procedure or retransmit reply</th>
<th>Invocation semantic</th>
<th>Used in …</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>Maybe</td>
<td>CORBA</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Re-execute procedure</td>
<td>At-least-once</td>
<td>Sun-RPC</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Retransmit old reply</td>
<td>At-most-once</td>
<td>RMI;CORBA</td>
</tr>
</tbody>
</table>

**Retransmit request message:** Whether or not to retransmit the request message until either a reply is received or the server is assumed to be failed.

**Duplicate filtering:** When retransmissions are used, whether to filter out duplicate requests at the server.

**Re-execute procedure or retransmit reply:** Whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations.
How do we implement the abstractions?

- What should the middleware do for making the call appear similar to a local call?
  
  - **Proxy** on the “client” (process P1) side
  - **Skeleton & dispatcher** on the “server” (process P2) side
Proxy

- Is responsible for making RMI transparent to clients by behaving like a local object to the invoker.
  - The proxy *implements* (Java term, not literally) the methods in the interface of the remote object that it represents. But,…

- Instead of executing an invocation, the proxy forwards it to a remote object.
  - On invocation, a method of the proxy *marshals* the following into a request message: (i) a reference to the target object, (ii) its own method id and (iii) the argument values. Request message is sent to the target, then proxy awaits the reply message, *un-marshals* it and returns the results to the invoker.
  - Invoked object unmarshals arguments from request message, and when done marshals return values into reply message.
Marshalling & Unmarshalling

• **External data representation**: an agreed, platform-independent, standard for the representation of data structures and primitive values.
  – CORBA Common Data Representation (CDR)
  – Allows a ARM client (possibly big endian) to interact with a x86 Unix server (little endian).

• **Marshalling**: the act of taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent).

• **Unmarshalling**: the process of disassembling data that is in external data representation form, into a locally interpretable form.
Remote Reference Module

- Is responsible for translating between local and remote object references and for creating remote object references.

- Has a *remote object table*
  - An entry for each remote object held by any process. E.g., B at P2.
  - An entry for each local proxy. E.g., proxy-B at P1.

- When a new remote object is seen by the remote reference module, it creates a remote object reference and adds it to the table.

- When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer to either a proxy or to a remote object.

- In case the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table.
How do we implement the abstractions?

- What should the middleware do for making the call appear similar to a local call?

- Proxy on the “client” (process P1) side
- **Skeleton & dispatcher** on the “server” (process P2) side
What about Server Side? Dispatcher and Skeleton

- Each process has one dispatcher, and a skeleton for each local object (actually, for the class).

- The dispatcher receives all request messages from the communication module.
  - For the request message, it uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message.

- Skeleton “implements” the methods in the remote interface.
  - A skeleton method un-marshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object).
  - It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message.
Proxy object is a hollow container of Method names.

Remote Reference Module translates between local and remote object references.

Dispatcher sends the request to Skeleton Object

Skeleton unmarshals parameters, sends it to the object, & marshals the results for return
SOAP – Simple Object Access Protocol

• Transmitted by HTTP or SMTP (or many others)
• Coded in XML (Can be decoded on any machine)
• Return value: Any XML document
• Underlies Web Services Description Language WSDL
Trade Example

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header/>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
JSON and REST
Hypertext Transfer Protocol (HTTP)

• A communications protocol

• Allows retrieving inter-linked text documents (hypertext)

• HTTP Verbs
  – HEAD
  – GET
  – POST
  – PUT
  – DELETE
  – TRACE
  – OPTIONS
  – CONNECT

```
Browser
```
```
Web Server
```

GET /index.html HTTP/1.1
Host: www.pitt.edu

HTTP/1.1 200 OK
Content-Type: text/html

<html><head>...
Representational State Transfer (REST)

• A style of software architecture for distributed hypermedia systems such as the World Wide Web.

• Introduced in the doctoral dissertation of Roy Fielding
  – One of the principal authors of the HTTP specification.

• A collection of network architecture principles which outline how resources are defined and addressed
REST and HTTP

• The motivation for REST was to capture the characteristics of the Web which made the Web successful.
  – URI Addressable resources
  – HTTP Protocol
  – Make a Request – Receive Response – Display Response

• Exploits the use of the HTTP protocol beyond HTTP POST and HTTP GET
  – HTTP PUT, HTTP DELETE
REST - not a Standard

- REST is not a standard
  - JSR 311: JAX-RS: The Java™ API for RESTful Web Services

- But it uses several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc (Resource Representations)
  - text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)
Main Concepts

Nouns (Resources)
unconstrained
i.e., http://example.com/employees/12345

Verbs
constrained
i.e., GET

Representations
constrained
i.e., XML

REST
Resources

• The key abstraction of information in REST is a resource.

• A resource is a conceptual mapping to a set of entities
  – Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on

• Represented with a global identifier (URI in HTTP)
  – http://www.boeing.com/aircraft/747
Naming Resources

• REST uses URI to identify resources

  – http://localhost/books/
  – http://localhost/classes
  – http://localhost/classes/cs2650
  – http://localhost/classes/cs2650/students

• As you traverse the path from more generic to more specific, you are navigating the data
Verbs

• Represent the actions to be performed on resources

• HTTP GET
• HTTP POST
• HTTP PUT
• HTTP DELETE
HTTP GET

• How clients ask for the information they seek.

• Issuing a GET request transfers the data from the server to the client in some representation

• GET http://localhost/books
  — Retrieve all books

• GET http://localhost/books/ISBN-0011021
  — Retrieve book identified with ISBN-0011021

• GET http://localhost/books/ISBN-0011021/authors
  — Retrieve authors for book identified with ISBN-0011021
HTTP PUT, HTTP POST

- HTTP POST creates a resource
- HTTP PUT updates a resource

- POST `http://localhost/books/`
  - Content: `{title, authors[], ...}`
  - Creates a new book with given properties

- PUT `http://localhost/books/isbn-111`
  - Content: `{isbn, title, authors[], ...}`
  - Updates book identified by isbn-111 with submitted properties
HTTP DELETE

• Removes the resource identified by the URI

Representations

• How data is represented or returned to the client for presentation.

• Two main formats:
  – JavaScript Object Notation (JSON)
  – XML

• It is common to have multiple representations of the same data
Representations

• **XML**
  
  – <COURSE>
    
    • <ID>CS2650</ID>
    
    • <NAME>Distributed Multimedia Software</NAME>
  
  – </COURSE>

• **JSON**
  
  – {course
    
    • {id: CS2650}
    
    • {name: Distributed Multimedia Software}
  
  – }
Architecture Style
Real Life Examples

- Google Maps
- Google AJAX Search API
- Yahoo Search API
- Amazon WebServices
Real Life: Flickr API

- **Resource:** Photos
- **Where:**
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_{mstb}.jpg
  - http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{o-secret}_o.(jpg|gif|png)
- **What:** JPEG, GIF or PNG (defined in the URL)
- **Example:** http://farm1.static.flickr.com/2/1418878_1e92283336_m.jpg
REST – Commands

You can associate commands with a resource.

Commands can replace the need for using HTTP methods and can provide a more familiar way of dealing with data.

Example:

```javascript
userResource = new Resource('http://example.com/users/001')
userResource.delete()
```
JSON – What is it?

• “JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate” – JSON.org

• Importantly: JSON is a subset of JavaScript
JSON – What does it look like?

```
{
  "firstName": "John",
  "lastName": "Smith",
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    "212 555-1234",
    "646 555-4567"
  ]
}
```
## JSON vs. XML

<table>
<thead>
<tr>
<th>JSON</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Structure</strong></td>
<td><strong>Data Structure</strong></td>
</tr>
<tr>
<td>No validation system</td>
<td>XSD</td>
</tr>
<tr>
<td>No namespaces</td>
<td>Has namespaces (can use multiples)</td>
</tr>
<tr>
<td>Parsing is just an eval</td>
<td>Parsing requires XML document parsing using things like XPath</td>
</tr>
<tr>
<td>• Fast</td>
<td></td>
</tr>
<tr>
<td>• Security issues</td>
<td></td>
</tr>
<tr>
<td>In JavaScript you can work with objects – runtime evaluation of types</td>
<td>In JavaScript you can work with strings – may require additional parsing</td>
</tr>
<tr>
<td>Security: Eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)</td>
<td>Security: XML is text/parsing – not code execution.</td>
</tr>
</tbody>
</table>
Server Load-balancing
Introduction to Load Balancing

- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server
Google: A Behind-the-Scenes Tour

Google Query Serving Infrastructure

Misc. servers
- Spell checker
- Ad Server

query

Google Web Server

Many corpora
- News
- Books
- ...

Index servers
- I₀, I₁, I₂, ..., Iₙ
- Index shards
- Replicas

Doc servers
- D₀, D₁, D₂, ..., Dₘ
- Doc shards
- Replicas

Elapsed time: 0.25s, machines involved: 1000s+
How do we split up information?
Information Strategies

Replication

Partition
## Load Balancing Approaches

<table>
<thead>
<tr>
<th>File Distribution</th>
<th>Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content/Locality Aware</td>
<td>DNS Server</td>
</tr>
<tr>
<td>Size Aware</td>
<td>Centralized Router</td>
</tr>
<tr>
<td>Workload Aware</td>
<td>Distributed Dispatcher</td>
</tr>
</tbody>
</table>
Issues

• Efficiently processing requests with optimizations for load balancing
  – Send and process requests to a web server that has files in cache
  – Send and process requests to a web server with the least amount of requests
  – Send and process requests to a web server determined by the size of the request
What does a SLB do?

- Gets user to needed resource:
  - Server must be available
  - User’s “session” must not be broken
    - If user must get to same resource over and over, the SLB device must ensure that happens (i.e., session persistence)
- In order to do work, SLB must:
  - Know servers – IP/port, availability
  - Understand details of some protocols (e.g., FTP, SIP, etc)
- Network Address Translation, NAT:
  - Packets are re-written as they pass through SLB device.
Why to Load-balance?

• Scale applications / services
• Ease of administration / maintenance
  – Easily and transparently remove physical servers from rotation in order to perform any type of maintenance on that server.
• Resource sharing
  – Can run multiple instances of an application / service on a server; could be running on a different port for each instance; can load-balance to different port based on data analyzed.
Load-Balancing Algorithms

• Most predominant:
  – least connections: server with fewest number of flows gets the new flow request.
  – weighted least connections: associate a weight / strength for each server and distribute load across server farm based on the weights of all servers in the farm.
  – round robin: round robin thru the servers in server farm.
  – weighted round robin: give each server ‘weight’ number of flows in a row; weight is set just like it is in weighted least least flows.

• There are other algorithms that look at or try to predict server load in determining the load of the real server.
How SLB Devices Make Decisions

• The SLB device can make its load-balancing decisions based on several factors.
  – Some of these factors can be obtained from the packet headers (i.e., IP address, port numbers, etc.).
  – Other factors are obtained by looking at the data beyond the network headers. Examples:
    • HTTP Cookies
    • HTTP URLs
    • SSL Client certificate
• The decisions can be based strictly on flow counts or they can be based on knowledge of application.
• For some protocols, like FTP, you have to have knowledge of protocol to correctly load-balance (i.e., control and data connection must go to same physical server).
When a New Flow Arrives

• Determine if virtual server exists.
  – If so, make sure virtual server has available resources.
  – If so, then determine level of service needed by that client to that virtual server.
    • If virtual machine is configured with particular type of protocol support of session persistence, then do that work.
  – Pick a real server for that client.
    • The determination of real server is based on flow counts and information about the flow.
    • In order to do this, the SLB may need to proxy the flow to get all necessary information for determining the real server – this will be based on the services configured for that virtual server.

• If not, the packet is bridged to the correct interface based on Layer 2.
SLB: Architectures

• Traditional
  – SLB device sits between the Clients and the Servers being load-balanced.

• Distributed
  – SLB device sits off to the side, and only receives the packets it needs to based on flow setup and tear down.
SLB: Traditional View with NAT
SLB: Traditional View without NAT

Client → SLB → Server1
Client → SLB → Server2
Client → SLB → Server3
Load-Balance: Layer 3 / 4

• Looking at the destination IP address and port to make a load-balancing decision.

• In order to do that, you can determine a real server based on the first packet that arrives.
Layer 3 / 4: Sample Flow

1: SYN
2: SLB makes decision on Server
3: SYN
4: SYN/ACK
5: SYN/ACK

Rest of flow continues through HTTP GET and Server response.
Load-Balance: Layer 5+

• The SLB device must terminate the TCP flow for an amount of time BEFORE the SLB decision can be made.

  – For example, the cookie value must be sent by the client, which is after the TCP handshake before determining the real server.
Layer 5+: Sample Flow

1: SYN
3: SYN/ACK
4: ACK
5: GET w/ Cookie

2: SLB device determines it must proxy flow before decision can be made.

Rest of flow continues with Server response.
Note: the flow can be unproxied at this point for efficiency.
FE: Forwarding Engines, which are responsible for forwarding packets. They ask the SLB device where to send the flow.
Distributed Architecture: Sample Flow

1: TCP SYN

Client → FE

2: FE asks where to send flow.

SE asks where to send flow.

SLB

3: Service Mgr tells it to use Server2.

FE → Server2

4: flow goes to Server2.

Server1

Subsequent packets flow directly from Client to Server2 thru the FE.
The FE must notify the SLB device when the flow ends.
Third Party Feedback: Network
RSerPool: Overview

- RSerPool protocols sit between the user application and the IP transport protocol (session layer).
- The application communication is now defined over a pair of logical session layer endpoints that are dynamically mapped to transport layer addresses.
- When a failure occurs at the network or transport layer, the session can survive because the logical session endpoints can be mapped to alternative transport addresses.
- The endpoint to transport mapping is managed by distributed servers providing resiliency.