A couple of new technologies

Worth getting to know

Neale Ferguson
Sine Nomine Associates
Topics

• ØMQ – Lightweight messaging
• CMIS – A Protocol for interacting with ECM systems
• Sones – GraphDB: a noSQL database
ØMQ

- “No man is an island”
- Many options
  - MQSeries
  - RabbitMQ
  - ApacheMQ
  - OpenMQ
- Many attributes
  - %CPU
  - Footprint
  - Latency
  - Configuration (brokers etc.)
ØMQ

- ZeroMQ
- Library of APIs
- Modeled on standard TCP/IP semantics
- Not a message broker
  - But can be used to create one
static size_t __inline__getData(int sd, char *buffer, size_t size)
{
    size_t lMsg = 0;
    while (lMsg < size) {
        lMsg = recv(sd, (buffer+lMsg), (size – lMsg));
    }
    return(lMsg);
}

size_t msgLen;
char *msgData;

getData(sd, (char *) &msgLen, sizeof(msgLen));
msgData = malloc(msgLen);
data(sd, msgData, msgLen);
ØMQ

```c
zmq_msg_t request;
char *msgData;
int msgLen;

zmq_msg_init(&request);
zmq_recv(sd, &request, 0);
msgLen = zmq_msg_size(&request);
msgData = zmq_msg_data(&request);
```
ØMQ

- Multiple Carriers
  - tcp:// is a plain old TCP socket with a host and port number.
  - ipc:// uses UNIX inter-process communication such as domain sockets, MQ, or whatever is available.
  - inproc:// is an in-process transport that passes messages via memory directly between threads sharing a single ØMQ context.
  - pgm:// is reliable multicast messaging that uses raw IP layering and requires special privileges.
  - epgm:// is an encapsulated version that uses regular User Datagram Protocol (UDP) to do reliable multicast messaging.
ØMQ

• N-to-N Dissemination
  • ØMQ sockets may be connected to multiple endpoints using `zmq_connect()`, while simultaneously accepting incoming connections from multiple endpoints bound to the socket using `zmq_bind()`. This allows many-to-many relationships

• Low Overhead and Fast Messaging

• Asynchronous I/O

• No need for mutexes, locks, or any other form of inter-thread communication
ØMQ

Diagram showing the relationship between Node and Socket with '1 to N Fan out' and 'N to 1 Fan in' connections.
ØMQ

- Language bindings exist for:
  - Ada
  - Basic
  - C
  - Chicken Scheme
  - Common Lisp
  - C# (.NET & Mono)
  - C++
  - D
  - Erlang
  - Go
  - Haskell
  - Java
  - Lua
  - node.js
  - Objective-C
  - ooc
  - Perl
  - PHP
  - Python
  - Racket
  - Ruby
  - Tcl

- ØMQ is available on multiple platforms, including Linux, Windows, Solaris, and OpenVMS.
### ØMQ

<table>
<thead>
<tr>
<th>Request-Reply</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ØMQ::REQ</strong></td>
<td>Used by a client to send requests to and receive replies from a service. Each request sent is load-balanced among all services, and each reply received is matched with the last issued request.</td>
</tr>
<tr>
<td><strong>ØMQ::REP</strong></td>
<td>Used by a service to receive requests from and send replies to a client.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>ØMQ::REQ</th>
<th>ØMQ::REP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible peer sockets</td>
<td>ZMQ::REP, ZMQ::XREP</td>
<td>ZMQ::REQ, ZMQ::XREQ</td>
</tr>
<tr>
<td>Direction</td>
<td>Bidirectional</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>Send/receive pattern</td>
<td>Send, Receive, Send, Receive, ...</td>
<td>Receive, Send, Receive, Send, ...</td>
</tr>
<tr>
<td>Outgoing routing strategy</td>
<td>Load-balanced</td>
<td>Fair-queued</td>
</tr>
<tr>
<td>Incoming routing strategy</td>
<td>Last peer</td>
<td>Last peer</td>
</tr>
<tr>
<td>HWM action</td>
<td>Block</td>
<td>Drop</td>
</tr>
</tbody>
</table>
ØMQ

Client

REQ

"Hello"

REP

"World"

Server
ØMQ

Diagram:
- REQ
  - ROUTER
    - code
    - DEALER
  - REP
- REQ
  - REP
- REQ
  - REP
<table>
<thead>
<tr>
<th><strong>Publish-Subscribe</strong></th>
<th><strong>Used for one-to-many distribution of data from a single publisher to multiple subscribers in a fan-out fashion</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMQ::PUB</td>
<td>Used by a publisher to distribute data. Messages sent are distributed in a fan-out fashion to all connected peers</td>
</tr>
<tr>
<td>Compatible peer sockets</td>
<td>ZMQ::SUB</td>
</tr>
<tr>
<td>Direction</td>
<td>Unidirectional</td>
</tr>
<tr>
<td>Send/receive pattern</td>
<td>Send only</td>
</tr>
<tr>
<td>Incoming routing strategy</td>
<td>N/A</td>
</tr>
<tr>
<td>Outgoing routing strategy</td>
<td>Fan-out</td>
</tr>
<tr>
<td>HWM action</td>
<td>Drop</td>
</tr>
<tr>
<td>ZMQ::SUB</td>
<td>Used by a <em>subscriber</em> to subscribe to data distributed by a <em>publisher</em>.</td>
</tr>
<tr>
<td>Compatible peer sockets</td>
<td>ZMQ::PUB</td>
</tr>
<tr>
<td>Direction</td>
<td>Unidirectional</td>
</tr>
<tr>
<td>Send/receive pattern</td>
<td>Receive only</td>
</tr>
<tr>
<td>Incoming routing strategy</td>
<td>Fair-queued</td>
</tr>
<tr>
<td>Outgoing routing strategy</td>
<td>N/A</td>
</tr>
<tr>
<td>ZMQ::HWM option action</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Pipeline</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>ØMQ</strong></td>
<td>Used for distributing data to <em>nodes</em> arranged in a pipeline. Data always flows down the pipeline, and each stage of the pipeline is connected to at least one <em>node</em>. When a pipeline stage is connected to multiple <em>nodes</em> data is load-balanced among all connected <em>nodes</em>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*<em>ZMQ::<em>PUSH</em></em></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compatible peer sockets</strong></td>
<td>ZMQ::*PULL</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>Unidirectional</td>
</tr>
<tr>
<td><strong>Send/receive pattern</strong></td>
<td>Send only</td>
</tr>
<tr>
<td><strong>Incoming routing strategy</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Outgoing routing strategy</strong></td>
<td>Load-balanced</td>
</tr>
<tr>
<td><strong>HWM action</strong></td>
<td>Block</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*<em>ZMQ::<em>PULL</em></em></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compatible peer sockets</strong></td>
<td>ZMQ::*PUSH</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>Unidirectional</td>
</tr>
<tr>
<td><strong>Send/receive pattern</strong></td>
<td>Receive only</td>
</tr>
<tr>
<td><strong>Incoming routing strategy</strong></td>
<td>Fair-queued</td>
</tr>
<tr>
<td><strong>Outgoing routing strategy</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>HWM action</strong></td>
<td>N/A</td>
</tr>
</tbody>
</table>
ØMQ
ØMQ

# pub.py - Publish weather data for multiple zipcodes
import zmq
import random

context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://*:5556")

while True:
    zipcode = random.randrange(10000,11000)
    temperature = random.randrange(1,215) - 80
    relhumidity = random.randrange(1,50) + 10

    socket.send("%d %d %d" % (zipcode, temperature, relhumidity))
# sub.py - Subscribe to weather data for a given zipcode
import sys
import zmq

context = zmq.Context()
socket = context.socket(zmq.SUB)

print "Collecting updates from weather server..."
socket.connect ("tcp://localhost:5556")

# Subscribe to zipcode, default is NYC, 10001
filter = sys.argv[1] if len(sys.argv) > 1 else "10001"
socket.setsockopt(zmq.SUBSCRIBE, filter)

# Process 5 updates
total_temp = 0
for update_nbr in range (5):
    string = socket.recv()
    zipcode, temperature, relhumidity = string.split()
    total_temp += int(temperature)

print "Average temperature for zipcode '%s' was %dF" % (filter, total_temp / update_nbr)
[neale@fedora ~]$ python pub.py &
[neale@fedora ~]$ python sub.py 10200 &
[neale@fedora ~]$ python sub.py 10300 &
[neale@fedora ~]$ python sub.py 10400 &
[neale@fedora ~]$ python sub.py 10500 &
Collecting updates from weather server...
Collecting updates from weather server...
Collecting updates from weather server...
Collecting updates from weather server...
Average temperature for zipcode '10400' was 16F
Average temperature for zipcode '10500' was 18F
Average temperature for zipcode '10200' was 30F
Average temperature for zipcode '10300' was 15F
ØMQ

Diagram showing network components with SNMP and OMQ labels.
CMIS
A Protocol for Accessing and manipulating ECM Systems
CMIS

- A specification for improving interoperability between Enterprise Content Management systems
- OASIS specification
- Participants include Liferay, Alfresco, eXo, Day Software, EMC, FatWire, IBM, Microsoft, Open Text, Oracle and SAP
CMIS

- Is language-agnostic (REST and SOAP are implemented in many languages)
- Decouples web service and content: CMIS can be used to access to an historic document repository
def CreateCmisFolderIfItDoesNotExist(targetFolderObject, newFolderName):
    #-----------------------------------------------------------#
    # first lets find out if a folder already exists by this    #
    # name (newFolderName)                                      #
    #-----------------------------------------------------------#
    children = targetFolderObject.getChildren()
    for child in children:
        if (child.name == newFolderName):
            return child
    logger.debug("Creating folder " + newFolderName)
    return targetFolderObject.createFolder(newFolderName)

props = createPropertyBag(propBag, targetClass)
f = open(docLocalPath, 'rb')
newDoc = folder.createDocument(docName, props, contentFile=f)
logger.debug("Cmislib create returned id=" + newDoc.id)
f.close()
Company Home  >  testdata

testdata
This view allows you to browse the items in this space.

Browse Spaces

SNAVM4_NEALE_20110210181814_0055
10 February 2011 13:20
16 Mb

Content Items

No items to display. To add an existing document click 'Add Content' action. To create an HTML or Plain
IELW2322I 1220 1  
IELW2322I 1220 2  
IELW2322I 1220 3  
IELW2322I 1220 4  
IELW2322I 1220 5  
IELW2322I 1220 6  
IELW2322I 1220 7  
IELW2322I 1220 8  
IELW2322I 1220 9  
IELW2322I 1220 10  
IELW2322I 1220 11  
IELW2322I 1220 12  
IELW2322I 1220 13  
IELW2322I 1220 14  
IELW2322I 1220 15  
IELW2322I 1220 16  
IELW2322I 1220 17  
IELW2322I 1220 18  
IELW2322I 1220 19  
IELW2322I 1220 20  
IELW2322I 1220 21  
IELW2322I 1220 5623  
IELW2322I 1220 5624  
IELW2322I 1220 5625  
IELW2322I 1220 5626  
IELW2322I 1220 5627  
IELW2322I 1220 5646  
IELW2617I 4C43 DEFINITION SIDE FILE IS EMPTY. THERE ARE NO SYMBOLS TO BE EXPORTED.

SHARE in Orlando
2011