Pararell and distribiuted programming

Author: Artur Skonecki
Scientific supervisor: Zuzanna Krawczyk

09/05/2013

Contents

1 Abstract **2**

2 Background **2**

3 Design, implementation , and test setup **2**
3.1 Data format 3
3.2 server.py 4
3.3 client.py 4
3.4 dbsupport.py 4

4 Installation **5**

5 Experiments **5**

6 Conclusion and future owrk **5**

7 Code **5**
7.1 client.py 5
7.2 server.py 8
7.3 dbsupport.py 8

1 Abstract

This report outlines the design and development of a computer software system for pararell XML xpath extraction from RSS feeds with support for a yet unknown database. This program was written in Python to run under the Unix operating system.

The design and ensuing program are modular in nature (server-client architecture) and make maximum use of abstract data types and of software re-use. Particular attention is paid to performance increase through pararellization Client-server architecture provides the ability to use implemented features from any other program.

The report includes a full user manual, as well as the whole of the code that was written. The source code was written with a particular focus on readability and clarity.

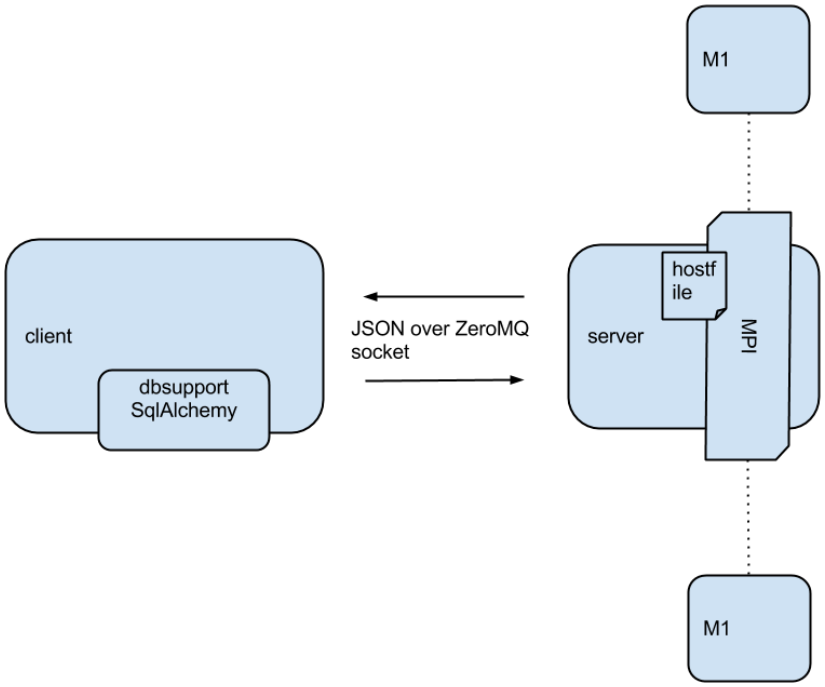
2 Background

Used technologies:

- MPI - Message Passing Interface
- PyZMQ - ZeroMQ bindings for Python
- SqlAlchemy - generalized database access
- JSON - JavaScript Object Notation

3 Design, implementation , and test setup

The architecture.



3.1 Data format

Format request sent by client to server:

```
{
"url":url,
"article_nums": article_nums,
"xpath": xpath
}
```

Format of answers sent by server to client:

```
Reply format:
{
"article number' : list of extracted items,
"article number' : list of extracted items,
...
}
```

3.2 server.py

A program serving extracts of contents of articles in rss feed over zmq sockets using json as data format. This implementation uses MPI for speeding up execution so it is taking advantage of concurrency features of modern systems.

```
server( port )
  Start a server listening for connections with zmq socket at 'port'
  for json requests from clients.

extract( xml, article_nums, xpath ) -> dict
  Return a dict containing article extracts.
```

3.3 client.py

An implementation of a client:

- request from server extracts of contents of articles in rss feed
- fetch the response
- write results to a dummy database:

  ```
  TExtract( url, xpath, contents ) |one-to-many| TContent( content )
  ```

- print out database

Uses json as data format. ZeroMQ is deployed for communication between client and server. SQLAlchemy for database access.

```
get_article_extracts( port, url, article_nums, xpath ) -> dict
  Return a dict containing rss article extracts.

main()
```

3.4 dbsupport.py

A file containing classes implementing access to databases through SqlAlchemy.

```
class TExtract(Base)

class TContent(Base)

class DbSupport( object )
```

4 Installation

Install required packages on all hosts.

```
apt-get install openmpi-bin libopenmpi-dev build-essentials python-dev python-zm
pip install mpi4py
```

If server is supposed to run on multiple machines create a hostfile where server.py will be started. The username should be the same on all machines. 4k2 directory should be the same on all machines.

```
root@voyage:~/4k2# cat ~/hostfile
192.168.0.17
192.168.0.19
```

5 Experiments

Start server on a machine. Include hostfile if running on multiple machines.

```
mpirun -np 4 --hostfile ~/hostfile python server.py
```

Extracting articles from a remote rss feed.

```
 python client.py  -f http://feeds.feedburner.com/TechCrunch  -n 2,5,6,9,10 -s c
```

Extracting articles from a file located on the filesystem

```
 python client.py  -f file://‘pwd‘/rss/TechCrunch.rss
-n 2,5,6,9,10 -s title
```

6 Conclusion and future owrk

Possible future directions:

- authentication

7 Code

7.1 client.py

```
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-
# python <3
```

```
# 2013 Artur Skonecki

"""
An implementation of a client:
- request from server extracts of contents of articles in rss feed
- fetch the response
- write results to a dummy database:
  TExtract( url, xpath, contents ) |one-to-many| TContent( content )
- print out database
Uses json as data format.
ZeroMQ is deployed for communication between client and server.
SQLAlchemy for database access.
"""

PORT = "5556"

import json
import zmq

from optparse import OptionParser

import dbsupport

# Connect to a server over zmq socket. Send a request for contents (xpath)
# from specific articles (article_nums) published on a rss feed (url).
# Fetch the reponse back.
def get_article_extracts( host, port, url, article_nums, xpath ):
  '''get_article_extracts( port, url, article_nums, xpath ) -> dict

  Return a dict containing rss article extracts.
  '''

  # connect to a server
  context = zmq.Context()
  socket = context.socket( zmq.REQ )
  socket.connect ( "tcp://%s:%s" % (host, port) )

  # format and send a json request over zmq socket
  jdata = json.dumps(
    "url":url,
    "article_nums": article_nums,
```

```
    "xpath": xpath
     )
  print( "Sending request " + str( jdata ) )
  socket.send( jdata )

  # get the reply and decode json
  message = socket.recv()
  json_decoder = json.JSONDecoder()
  jdata_reply = json_decoder.decode( message )

  return jdata_reply

def main():
  '''main()'''
  parser = OptionParser(
    usage = 'Usage: python client.py  -f http://feeds.feedburner.com/TechCrunch

  parser.add_option( "-H", None,
    action="store",
    dest="host",
    default="localhost" )
  parser.add_option( "-f", None,
    action="store",
    dest="url",
    default="http://feeds.feedburner.com/TechCrunch" )
  parser.add_option( "-n", None,
    action="store",
    dest="article_nums",
    default="1,2,3" )
  parser.add_option( "-s", None,
    action="store",
    dest="xpath",
    default = 'category' )

  options = parser.parse_args()[0]

  extracts = get_article_extracts( options.host, PORT,
    options.url,
    [ int( x ) for x in options.article_nums.split( ',' ) ],
    options.xpath )
```

```
  dba = dbsupport.DbSupport( 'sqlite:///:memory:' )

  dba.write( options.url,
    options.xpath,
    extracts )

  dba.print_db()

if __name__ == '__main__':
  main()
```

7.2 server.py

\input{../server.py}

7.3 dbsupport.py

```
# -*- coding: utf-8 -*-
# python <3
# 2013 Artur Skonecki

'''
A file containing classes implementing access to databases through SqlAlchemy
'''

from sqlalchemy import *
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relation, sessionmaker, relationship, backref

Base = declarative_base()

class TExtract(Base):
  __tablename__ = 'extracts'

  id = Column(Integer, primary_key=True)
```

```
  url = Column(String(255), nullable=False)
  xpath = Column(String(255), nullable=False)
  contents = relationship("TContent", backref="extracts")

  def __init__(self, url=None, xpath=None, contents=None):
    self.url = url
    self.xpath = xpath
    for item in contents:
      self.contents.append( TContent( item ) )
  def __repr__(self):
    return "TExtract(%r, %r, %r)" % ( self.url, self.xpath, self.contents )

class TContent(Base):
  __tablename__ = 'contents'
  cid = Column(Integer, primary_key=True)
  parent_id = Column(Integer, ForeignKey('extracts.id'))
  content = Column(String(1023))

  def __init__(self, content=None):
    self.content = content

  def __repr__(self):
    return "TContent(%r)" % ( self.content )

class DbSupport( object ):

  def __init__( self, dba ):
    '''Contruct a new ``DbSupport`` object

    :param dba: specify database for SqlAlchemy
      e.g.:
        DbSupport( 'sqlite:///:memory:' )
    '''
    engine = create_engine( dba )
    Base.metadata.create_all( engine )

    Session = sessionmaker(bind=engine)
    self.session = Session()


  def write( self, url, xpath, extracts ):
```

```
  '''Write records to database'''
  try:
    for content in extracts.itervalues():
      print content
      self.session.add( TExtract( url, xpath, content ) )
    self.session.commit()
  except:
    self.session.rollback()
    raise

def print_db( self ):
  '''Print out all TExtract records'''
  alldata = self.session.query(TExtract).all()
  for data in alldata:
    print( data )
```