法律声明

- □ 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- □ 课程详情请咨询
  - ■ 微信公众号：大数据分析挖掘
  - ■ 新浪微博：ChinaHadoop

分布式爬虫

大纲

- 分布式系统概述
- 主从服务设计

分布式系统

Deduce of Distributed System - I

- A *program*

  is the code you write.

- A *process*

  is what you get when you run it.

- A *message*

  is used to communicate between processes.

- A *packet*

  is a fragment of a message that might travel on a wire.

- A *protocol*

  is a formal description of message formats and the rules that two processes must follow in order to exchange those messages.

Distributed System - II

- A *network*

  is the infrastructure that links computers, workstations, terminals, servers, etc. It consists of routers which are connected by communication links.

- A *component*

  can be a process or any piece of hardware required to run a process, support communications between processes, store data, etc.

- A *distributed system*

  is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network, such that all components cooperate together to perform a single or small set of related tasks.

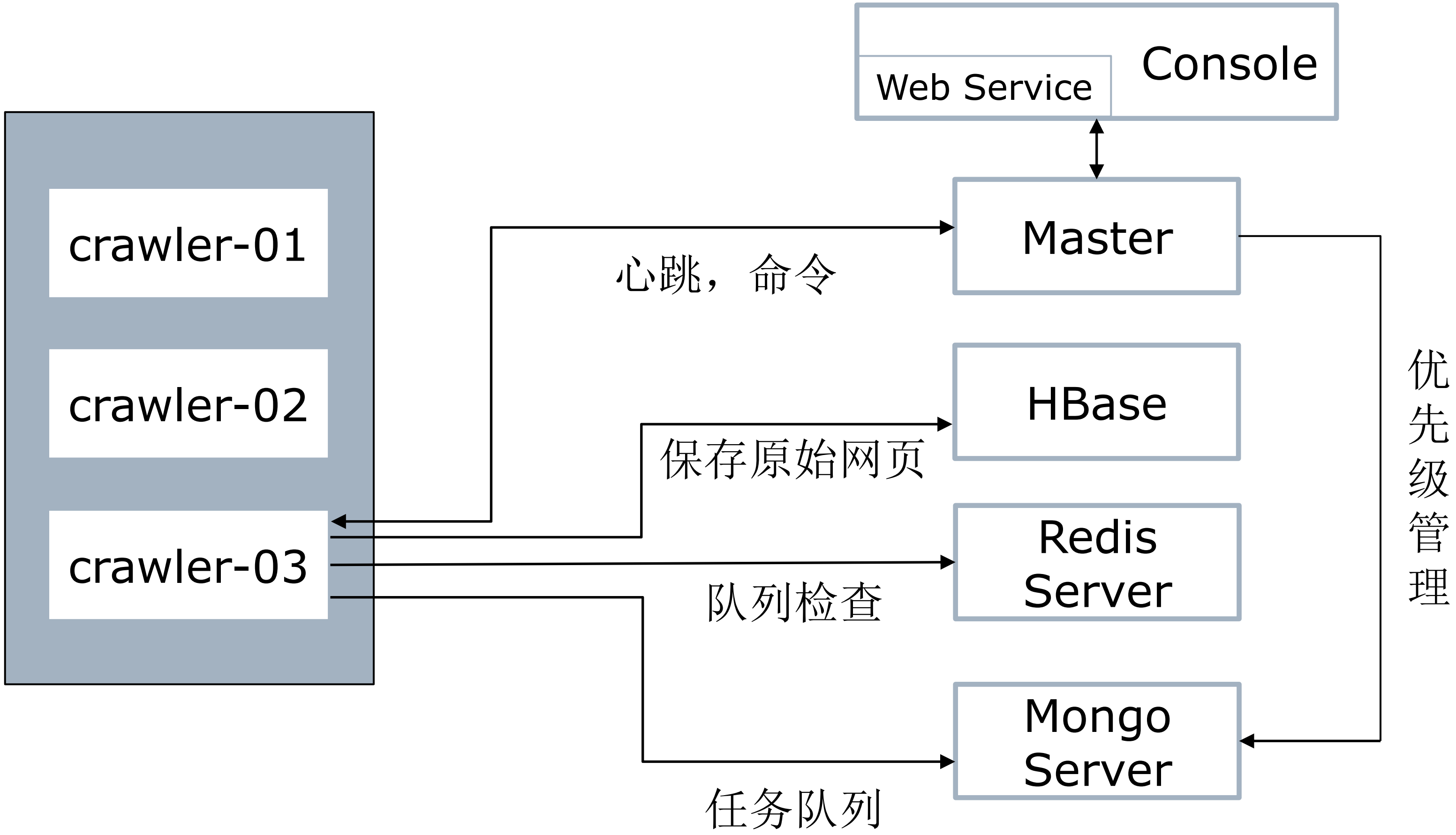
Advantage

- Fault-Tolerant: It can recover from component failures without performing incorrect actions.
- Highly Available: It can restore operations, permitting it to resume providing services even when some components have failed.
- Recoverable: Failed components can restart themselves and rejoin the system, after the cause of failure has been repaired.
- Consistent: The system can coordinate actions by multiple components often in the presence of concurrency and failure. This underlies the ability of a distributed system to act like a non-distributed system.
- Scalable: It can operate correctly even as some aspect of the system is scaled to a larger size.
- Predictable Performance: The ability to provide desired responsiveness in a timely manner.
- Secure: The system authenticates access to data and services

Challenge

- Replications and migration cause need for ensuring consistency and distributed decision-making
- Failure modes: Not assuming data received is same as sent
- Concurrency: Update/Replication/Cache/Failure …
- Heterogeneity: Network, hardware, OS, languages, developers
- Scalability: Architecture must be able to handle increase of users, resources, etc. Considering cost of physical resources, performance loss, bottleneck
- Security

分布式爬虫系统


Console
Web Service
crawler-01
crawler-02
crawler-03
Master
心跳，命令
HBase
保存原始网页
Redis Server
队列检查
Mongo Server
任务队列
优先级管理

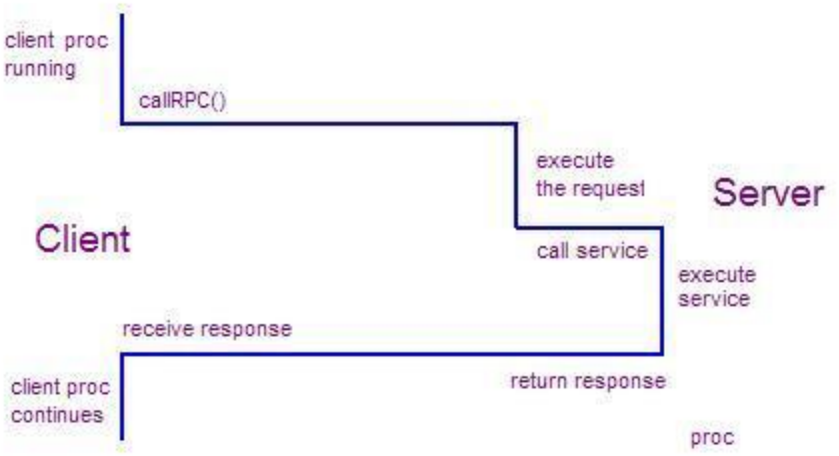
Master-Slave 结构

Master-Slave 结构

- 有一个主机，对所有的服务器进行管理。绝大多数分布式系统，都是 Master-Slave 的主从模式。而之前我们的爬虫，是完全独立的，依次从url队列里获取url，进行抓取
- 当爬虫服务器多的时候，必须能通过一个中心节点对从节点进行管理
- 能对整体的爬取进行控制
- 爬虫之间信息共享的桥梁
- 负载控制

Remote Procedure Calls

- Specifies the protocol for client-server communication
- Develops the client program
- Develops the server program



Protocol – Message Type

```
# message type, REGISTER, UNREGISTER and HEARTBEAT
MSG_TYPE      = 'TYPE’
# send register
REGISTER      = 'REGISTER’
# unregister client with id assigned by master
UNREGISTER    = 'UNREGISTER’
# send heart beat to server with id
HEARTBEAT     = 'HEARTBEAT’
# notify master paused with id
PAUSED        = 'PAUSED’
# notify master resumed with id
RESUMED       = 'RESUMED’
# notify master resumed with id
SHUTDOWN      = 'SHUTDOWN’
```

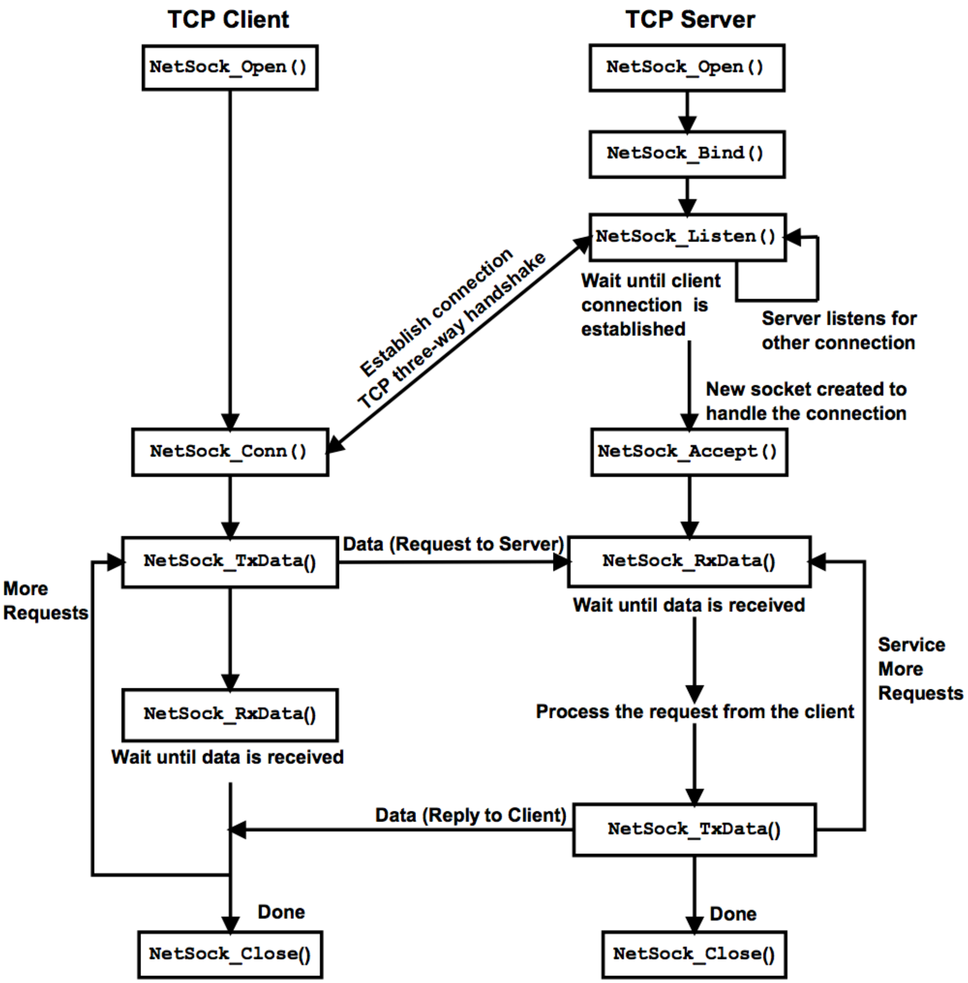
Protocol - Actions

```
# server status key word
ACTION_REQUIRED       = 'ACTION_REQUIRED’
# server require pause
PAUSE_REQUIRED        = ‘PAUSE_REQUIRED’
# server require pause
RESUME_REQUIRED       = 'RESUME_REQUIRED’
# server require shutdown
SHUTDOWN_REQUIRED           = 'SHUTDOWN_REQUIRED'
```

Protocol – Key Definition

```
# server status key word
SERVER_STATUS           = SERVER_STATUS
# client id key word
CLIENT_ID               = ‘CLIENT_ID’
# error key work
ERROR                   = ERROR’
```

Socket



TCP Client
TCP Server
NetSock_Open()
NetSock_Open()
NetSock_Bind()
NetSock_Listen()
Wait until client connection is established
Server listens for other connection
Establish connection
TCP three-way handshake
New socket created to handle the connection
NetSock_Conn()
NetSock_Accept()
Data (Request to Server)
NetSock_TxData()
NetSock_RxData()
More Requests
Wait until data is received
Service More Requests
Process the request from the client
NetSock_RxData()
Wait until data is received
Data (Reply to Client)
NetSock_TxData()
Done
Done
NetSock_Close()
NetSock_Close()

Create Client Socket

```
#create an INET, STREAMing socket
s = socket.create_connection( socket.AF_INET,
                                          socket.SOCK_STREAM)
```

AF_INET -- IPv4 Internet protocols

SOCK_STREAM, SOCK_DGRAM, SOCK_RAW -- socket types (SOCK_STREAM TCP, SOCK_DGRAM UDP)

Create Server Socket

```
#create an INET, STREAMing socket
serversocket = socket.socket( socket.AF_INET,
                          socket.SOCK_STREAM)
#bind the socket to a public host, and a well-known port
serversocket.bind((socket.gethostname(), 20010))
#become a server socket
serversocket.listen(5)
```

listen(backlog) -- number of unaccepted connections that the system will allow before refusing new connections, at least 0

Create Server Socket

```
while True:
        #accept connections from outside
         (clientsocket, address) = serversocket.accept()
        #now do something with the clientsocket
        #in this case, we'll pretend this is a threaded
        server ct = client_thread(clientsocket)
        ct.run()
```

Ways to listening

- a new thread to handle clientsocket
- a new process
- use non-blocking socket

Non-blocking mode listening

- connection.setblocking(False),

  send, recv, connect and accept returns immediately

  connection.setblocking(False) is equivalent to settimeout(0.0)

- asyncore

  Provides the basic infrastructure for writing asynchronous

  socket service clients and servers.

Event	Description
handle_connect()	Implied by the first read or write event
handle_close()	Implied by a read event with no data available
handle_accept()	Implied by a read event on a listening socket

Ways to end communication

- fixed length message: **while** totalsent < MSGLEN:
- delimited: some message**\0**
- indicates message length in beginning: LEN: 50;
- shutdown connection: server call close(), clietn recv() returns 0

疑问

- 问题答疑：　　http://www.xxwenda.com/
  - 可邀请老师或者其他人回答问题

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop