Pig Laboratory

This laboratory is dedicated to Hadoop Pig and consists of a series of exercises: some of them somewhat mimic those in the MapReduce laboratory, others are inspired by "real-world" problems.

The main goal for this laboratory is to gain familiarity with the Pig Latin language to analyze data in many different ways. In other words, to focus on "what to do" with your big data: to perform some statistics, to mine out useful information, or to implement some simple algorithms, etc... This is a typical work for a "data scientist".

The interested student can understand the details of Hadoop Pig internals by inspecting the process of turning a Pig Latin script into a runnable, optimized underlying implementation in MapReduce. To this aim, the students should examine what the Pig compiler generates from a Pig Latin script (using EXPLAIN command), and reason about Hadoop Job performance by analyzing Hadoop logs and statistics. The EXPLAIN command can generate .dot files that illustrate the DAG (directed acyclic graph) of the MapReduce jobs produced by Pig, and can be visualized by some graph-chart tools, such as GraphViz.

Additional documentation for the laboratory

The underlying assumption is that students taking part to this laboratory are familiar with MapReduce and Pig/Pig Latin. Additional documentation that is useful for the exercises is available here: http://pig.apache.org/docs/r0.11.0/
Note that we will use Hadoop Pig 0.11.0, included in the Cloudera distribution of Hadoop, CDH 4.4.0.

Exercises and Rules

The general rule when using a real cluster is the following:

- First work locally: pig -x local: you can use both the interactive shell or directly work on pig scripts, to operate on data residing in the local filesystem
- Then submit job to the cluster: pig -x mapreduce NOTE : remember that a script that works locally may require some minor modifications when submitted to the Hadoop cluster. For example, you may want to explicitly set the degree of parallelism for the "reduce" phase, using the PARALLEL clause.
- If you're using the virtual machine prepared for the class, note that, each VM has its own pseudo-local installation of Hadoop, so you do not strictly need to use the “local” pig environment.

Tstat Data

Exercises are based on traces in Tstat format (see http://tstat.tlc.polito.it/index.shtml).

Tstat produces two files, "log_tcp_complete" and "log_tcp_nocomplete" files which log every TCP connection that has been tracked. A TCP connection is identified when the first SYN segment is observed, and is ended when either:

- the FIN/ACK or RST segments are observer;
- no data packet has been observed (from both sides) for a default timeout of 10 s after the three-way handshake or 5 min after the last data packet.

Tstat discards all the connections for which the three way handshake is not properly seen. Then, in case a connection is correctly closed, it is stored in log_tcp_complete, otherwise in log_tcp_nocomplete.

In the following exercise we will use a file called "tstat-big.txt”, which contains only correctly closed connections. The file consists of a line per each TCP connection; each line consists of fields, separated by spaces. Columns are grouped according to C2S - Client-to-Server and S2C - Server-to-Client traffic directions. The exact TSTAT file format is reported in the table immediately below.

C2S	S2C	Short Description	Unit	Long Description
1	45	Client/Server IP addr	-	IP addresses of the client/server
2	46	Client/Server TCP port	-	TCP port addresses for the client/server
3	47	packets	-	total number of packets observed form the client/server
4	48	RST sent	0/1	0 = no RST segment has been sent by the client/server
5	49	ACK sent	-	number of segments with the ACK field set to 1
6	50	PURE ACK sent	-	number of segments with ACK field set to 1 and no data
7	51	unique bytes	bytes	number of bytes sent in the payload
8	52	data pkts	-	number of segments with payload
9	53	data bytes	bytes	number of bytes transmitted in the payload, including retransmissions
10	54	rexmit pkts	-	number of retransmitted segments
11	55	rexmit bytes	bytes	number of retransmitted bytes
12	56	out seq pkts	-	number of segments observed out of sequence
13	57	SYN count	-	number of SYN segments observed (including rtx)
14	58	FIN count	-	number of FIN segments observed (including rtx)
15	59	RFC1323 ws	0/1	Window scale option sent
16	60	RFC1323 ts	0/1	Timestamp option sent
17	61	window scale	-	Scaling values negotiated [scale factor]
18	62	SACK req	0/1	SACK option set
19	63	SACK sent	-	number of SACK messages sent
20	64	MSS	bytes	MSS declared
21	65	max seg size	bytes	Maximum segment size observed
22	66	min seg size	bytes	Minimum segment size observed
23	67	win max	bytes	Maximum receiver window announced (already scale by the window scale factor)
24	68	win min	bytes	Maximum receiver windows announced (already scale by the window scale factor)
25	69	win zero	-	Total number of segments declaring zero as receiver window
26	70	cwin max	bytes	Maximum in-flight-size computed as the difference between the largest sequence number so far, and the corresponding last ACK message on the reverse path. It is an estimate of the congestion window
27	71	cwin min	bytes	Minimum in-flight-size
28	72	initial cwin	bytes	First in-flight size, or total number of unack-ed bytes sent before receiving the first ACK segment
29	73	Average rtt	ms	Average RTT computed measuring the time elapsed between the data segment and the corresponding ACK
30	74	rtt min	ms	Minimum RTT observed during connection lifetime
31	75	rtt max	ms	Maximum RTT observed during connection lifetime
32	76	Stdev rtt	ms	Standard deviation of the RTT
33	77	rtt count	-	Number of valid RTT observation
34	78	ttl_min	-	Minimum Time To Live
35	79	ttl_max	-	Maximum Time To Live
36	80	rtx RTO	-	Number of retransmitted segments due to timeout expiration
37	81	rtx FR	-	Number of retransmitted segments due to Fast Retransmit (three dup-ack)
38	82	reordering	-	Number of packet reordering observed
39	83	net dup	-	Number of network duplicates observed
40	84	unknown	-	Number of segments not in sequence or duplicate which are not classified as specific events
41	85	flow control	-	Number of retransmitted segments to probe the receiver window

C2S	S2C	Short Description	Unit	Long Description
42	86	unnece rtx RTO	-	Number of unnecessary transmissions following a timeout expiration
43	87	unnece rtx FR	-	Number of unnecessary transmissions following a fast retransmit
44	88	!= SYN seqno	0/1	1 = retransmitted SYN segments have different initial seqno
89		Completion time	ms	Flow duration since first packet to last packet
90		First time	ms	Flow first packet since first segment ever
91		Last time	ms	Flow last segment since first segment ever
92		C first payload	ms	Client first segment with payload since the first flow segment
93		S first payload	ms	Server first segment with payload since the first flow segment
94		C last payload	ms	Client last segment with payload since the first flow segment
95		S last payload	ms	Server last segment with payload since the first flow segment
96		C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
97		S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
98		First time abs	ms	Flow first packet absolute time (epoch)
99		C Internal	0/1	1 = client has internal IP, 0 = client has external IP
100		S Internal	0/1	1 = server has internal IP, 0 = server has external IP
101		Connection type	-	Bitmask stating the connection type as identified by TCPL7 inspection engine (see protocol.h)
102		P2P type	-	Type of P2P protocol, as identified by the IPP2P engine (see ipp2p_tstat.h)
103		P2P subtype	-	P2P protocol message type, as identified by the IPP2P engine (see ipp2p_tstat.c)
104		ED2K Data	-	For P2P ED2K flows, the number of data messages
105		ED2K Signaling	-	For P2P ED2K flows, the number of signaling (not data) messages
106		ED2K C2S	-	For P2P ED2K flows, the number of client<->server messages
107		ED2K C2C	-	For P2P ED2K flows, the number of client<->client messages
108		ED2K Chat	-	For P2P ED2K flows, the number of chat messages
109		HTTP type	-	For HTTP flows, the identified Web2.0 content (see the http_content enum in struct.h)
110		SSL Client Hello	-	For SSL flows, the server name indicated by the client in the Hello message extensions
111		SSL Server Hello	-	For SSL flows, the subject CN name indicated by the server in its certificate
112		Dropbox ID	-	Dropbox identifier of user device
113		FQDN	-	Full Qualified Domain Name contacted

1 Exercises

1.1 Exercise 1 – A "Network" Word Count

Problem statement: count the number of TCP connection per each client IP.

The problem is very similar to the Word Count problem of the MapReduce laboratory: it has been conceived to familiarize with PIG Latin.

Writing your first Pig Latin script

The following lines of code can also be submitted to the interactive pig shell (`grunt`) with some minor modifications. The code of this exercise is included in the archive on the website. For convenience, the code is also reported immediately below:

```
-- Load input data from local input directory
A = LOAD 'tstat-sample.txt' using PigStorage(' ') AS (ip_c:chararray, ….);


-- Group by client IP
B = GROUP A BY ip_c;


-- Generate the output data
C = FOREACH B GENERATE group, COUNT(A);


-- Store the output (and start to execute the script)
STORE C INTO 'output/ex1';
```

As you can notice, this exercise is solved (to be precise, this is one possible solution). You have the freedom to develop your own solutions. Using all the information we have provided so far, you will have to play with Pig and answer several questions at the end of this exercise (remember: Google may be a helpful friend, and so are we, so feel free to ask!). Some of the useful debugging commands are:

- **DESCRIBE** relation: this is very useful to understand the schema applied to each relation. Note that understanding schema propagation in Pig requires some time.
- **DUMP** relation: this command is similar to the STORE command, except that it outputs on `stdout` the selected relation.
- **ILLUSTRATE** relation: this command is useful to get a sample of the data in a relation.

Executing the script

Now that you are ready to submit you first pig script to the cluster, you need to specify the execution mode:

```
pig -x mapreduce script_name
```

When you interact with HDFS (e.g., when you create an output file) you will see a directory corresponding to your unix credentials (login) under the /user/ directory. Note that the pig script you wrote for local execution requires some modifications to be run on the cluster:

- Change the input path to the appropriate one. You can browse the local HDFS filesystem in order to find the tstat input files.
- Change the output path: note that Hadoop refuses to overwrite any local directory when you define the path, so, remember to use a different name or to delete the output directory for each run of your script.

- Set parallelism where appropriate, using the PARALLEL keyword. This is intentionally left for the student You can also use another troubleshooting instrument to understand the logical, physical and execution plans produced by Pig:
- EXPLAIN relation: note that you can generate output files in the "dot" format for better rendering How-to inspect your results and check your job in the cluster
- Inspecting your job status on the cluster: you can identify your job by name and check its status using the Web UI of the JobTracker so far. Make sure to obtain useful information, for example, what optimization Pig brings in to help reducing the I/O throughput.

Questions: 1. What does a GROUP BY command do? At which phase of MapReduce is GROUP BY performed in this exercise and in general? 2. What does a FOREACH command do? At which phase of MapReduce is FOREACH performed in this exercise and in general? 3. Explain very briefly how Pig works (i.e. the process of Pig turning a Pig Latin script into runnable MapReduce job(s)) 4. Explain how you come to a conclusion that your results are correct 5. How many reducers were launched? Why? Can you modify this number? 6. How many mappers were launched? Why? Can you modify this number?

1.2 Exercise 2

Problem statement: count the total number of TCP connection having “google.it” in the FQDN (Fully Qualified Domain Name) field (field #113 in the tstat data).

Hint: You need to modify the code of the previous exercise, filtering the loaded data, and applying a different grouping.

Questions: 1. How many reducers were launched? Can you increase the number of reducers?

1.3 Exercise 3

Problem statement: for each client IP, compute the sum of uploaded, downloaded and total (up+down) transmitted bytes.

1.4 Exercise 4

Problem statement: find the top 100 users per uploaded bytes.

Hint: there are different ways of solving this exercise, in particular, in recent PIG versions there is a function called TOP, that returns the top-n tuples from a bag of tuples (see http://pig.apache.org/docs/r0.11.1/func.html#topx).

Questions: 1. Is this job map-only? Why? Why not? 2. Where did you apply the TOP function? 3. Can you explain how does the TOP function work? 4. TOP function was introduced in PIG v.0.8. How, in your opinion, and based on your understanding of PIG, was the query answered before the TOP command was available? Do you think that it was less efficient than the current implementation?

1.5 Exercise 5

Problem statement: For each IP in the top 100 list previously computed, find the number of bytes uploaded by the largest TCP connection and the percentage of the bytes uploaded by this connection over the total number of uploaded bytes.

Hint: for this exercise, you need to join two datasets: the output of the previous exercise with the original tstat dataset.

Questions: 1. How many jobs were generated? 2. Describe how the join is executed.

1.6 Exercise 6

Problem statement: find the minimum value of client MSS (mss_c).

Questions: 1. Can you explain why did you find this value? 2. What did you learn from this exercise?

1.7 Exercise 7

Problem statement: find the percentage of TCP connections with minimum client window (win_min_c) == 1460 over the total number of connections.

Questions: 1. How many MR jobs were generated by PIG? 2. How many reducers were launched per each job? (did you use the PARALLEL keyword?)

1.8 Exercise 8

Problem statement: calculate the percentage of bytes received by each distinct server port over the total number of bytes received by all the server ports.

Questions: 1. How many reducers were launched? Which degree of parallelism did you choose? Why? 2. If you run the exercise on a large cluster, using the same dataset, would you use a different value of parallelism?

1.9 Exercise 9

Problem statement: Find the percentage of flows directed to port 80 over the total number of flows.

Questions: 1. Using the result of this exercise and the previous one, what can you say about the distribution of server ports?