

15-440/15-640: Homework 3

Due: November 10, 2016 10:30am

Name:

Andrew ID:

1 Hadoop (12 points)

1. Harry Bovik has just got access to a fancy GHC cluster (100 nodes) and decides to utilize every last resource available by launching a massive Hadoop job that spawns thousands of Map and Reduce tasks. However, to his disappointment, he finds that this job takes longer time to complete than he had expected from this cluster. He is later informed by the cluster admin that a couple of nodes use Intel Xeon processors that were released all the way back in 2007 and need to be upgraded to the latest Skylake Xeons that the rest of the nodes use. Why does he observe this issue, even though just 2 out of the 100 nodes in the cluster are sluggish? [4 points]

Hadoop is a batch-processing system that launches multiple Map and Reduce tasks in parallel that are scheduled across the nodes in a cluster and completes only when every single task of the job has completed. Even if all but 1 task finish in time T and 1 task requires time $2T$ to complete, the job's execution time is $2T$. This is the **tail latency problem** and tasks scheduled on the slower nodes act as **stragglers** that increase the overall execution time.

2. Harry has no time to wait for this hardware upgrade to happen and uses the knowledge he gained from 15-440 to arrive at a couple of solutions that he can directly implement on Hadoop. What are they? [4 points]

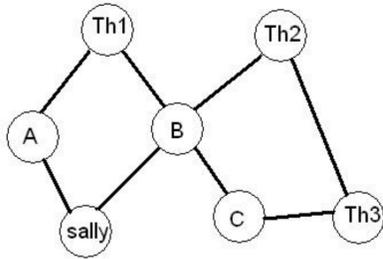
Hadoop has two ways of dealing with stragglers - (1) Detect and abort stragglers and schedule the task onto a different worker (2) **Speculative execution** - copy the task running on a straggler on another worker (if available). Take the result from the one that finishes first and kill the other.

3. After using the above solutions, Harry is still disappointed with the performance of his job. He profiles the cluster and discovers that the network I/O between HDFS nodes is the biggest bottleneck. He understands the problem and decides to write his own Hadoop scheduler. What was the problem and why does Harry make this decision? [4 points]

High volume of Network I/O between HDFS nodes implies that the data required by the Map and Reduce tasks aren't stored in the node on which they are scheduled. This would require the data to be transferred across the network from the source node. The solution for this problem is to maintain **data locality** - schedule tasks on the nodes that contain the data that they require. The default Hadoop scheduler uses a best-effort approach towards providing data locality to tasks. Since Harry wants to improve the performance, he decides to write his own scheduler tuned to his application's needs by offering better data locality guarantees than the default scheduler.

2 Iterative MapReduce (20 Points)

Sally has a very very very large map. She's really excited about finding all the movie theatres on the map, and how long it would take to get to each one (she collects movie tickets from various theatres, it's quite an impressive collection). Sally will need your help to find the shortest path from her house to every movie theatre using MapReduce. Assume the following: - There are arbitrary points on the graph that represent intersections, known as A, B, C, etc. - All edges on the graph are undirected. Here is an example of small map:

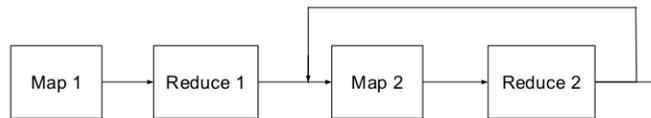


Let the above graph be represented by the following adjacency list -

```
Sally A 3
Sally B 5
Th1 A 1
B Th1 4
B Th2 6
C B 1
C Th3 4
Th2 Th3 2
```

Clearly, on a graph of smaller size, Sally can use Dijkstra's to find the information she needs, but because this graph is so large, she wants to use MapReduce to run a parallelized single-source shortest path algorithm. She designs a algorithm that consists of two phases

- The first phase reads and processes the input adjacency list of the graph (in the same format as shown above)
- The second phase receives the processed output from the first phase and runs multiple iterations of the *same* Map and Reduce code to finally output the shortest path to each theatre from Sally's location once the algorithm converges.



Hints -

- Trace the possible steps using this sample adjacency list input. Note that the output of a Reduce phase is the input for the subsequent Map phase.

- The values in <Key, Value> pairs for the Map input/output aren't limited to discrete data types (like int, floats, strings, etc). Permitted complex data types are lists and tuples and nestings of these two data types.
 - The input for Reduce is always a key followed by the list of all values emitted by Map for that key.
1. What would be the steps in the first Map phase? (Pseudo-code or high-level design logic will suffice) [2 Points]

In Phase 1 you accept the input adjacency list. Since the links are bi-directional, we emit both arrangements of end-vertices with the corresponding edge length.

2. What are the inputs and outputs from the first Map phase? (express as <Key, Value>.) [3 Points]

Phase 1 would do the following:

```
map:
input: <node1, (node2, distance)>
output: <node1, (node2, distance)>
<node2, (node1, distance)>
```

From the given example,

input to Phase 1 map:

```
Sally A 3
Sally B 5
Th1 A 1
B Th1 4
B Th2 6
C B 1
C Th3 4
Th2 Th3 2
```

```
output of map:
<Sally, (A, 3)>
<A, (Sally, 3)>
<Sally, (B, 5)>
<B, (Sally, 5)>
<Th1, (A, 1)>
<A, (Th1, 1)>
<B, (Th1, 4)>
<Th1, (B, 4)>
...
```

3. What would be the steps in the first Reduce phase? (Pseudo-code or high-level design logic will suffice) [2 Points]

The first Reduce phase performs "Dijkstra's initialization" - Sally's distance is set to 0 and every other node's distance is set as infinity.

4. What are the inputs and outputs from the first Reduce phase? (express as <Key,List of Values>.)
[3 Points]

Phase 1 Reduce would do the following:

```
reduce:
input: <node, {(node1, dist1), (node2, dist2), (node3, dist3)}>
output:
if(node is sally):
<node, {0, (node1, dist1), (node2, dist2), (node3, dist3)}>
else:
<node, {inf, (node1, dist1), (node2, dist2), (node3, dist3)}>
```

From the given example,

```
input to Phase 1 reduce:
<Sally, {(A, 3), (B, 5)}>
<B, {(Th1, 4), (Th2, 6), (Sally, 5), (C,1)}>
...
output of reduce:
<Sally, {0, (A, 3), (B, 5)}>
<B, {inf, (Th1, 4), (Th2, 6), (Sally, 5), (C,1)}>
```

5. What would be the steps in the second Map phase? (Pseudo-code or high-level design logic will suffice) [2 Points]

In Phase 2 we use the current shortest distance of each node from Sally and update the distance of its adjacent nodes from Sally.

6. What are the inputs and outputs from the second Map phase? (express as <Key,Value>.)
[3 Points]

```
map:
input: <node, {currdist, (node1, dist1), (node2, dist2), (node3, dist3)}>
output:
<node, {currdist, (node1, dist1), (node2, dist2), (node3, dist3)}>
if(currdist is inf):
ignore
else:
<node1, {(currdist+dist2)}>
<node2, {(currdist+dist2)}>
<node3, {(currdist+dist3)}>
```

From the given example,

```
input to Phase 2 map:
(the output of the reducer of phase 1)
output:
<Sally, {0, (A, 3), (B, 5)}>
<A, {3}>
<B, {5}>
<B, {inf, (Th1, 4), (Th2, 6), (Sally, 5), (C,1)}>
<Th1, {inf, (A, 1), (B,4)}>
...
```

7. What would be the steps in the second Reduce phase? (Pseudo-code or high-level design logic will suffice) [2 Points]

For each node we receive 2 sets of values, individual integers and a single list containing the adjacent nodes and current distance. The individual integers represent all possible distance values from Sally computed for this node from previous iterations. We find the minimum of these distance and update our current distance as this. Note: This design can be trivially extended to even capture the shortest path so far.

8. What are the inputs and outputs from the first Reduce phase? (express as <Key,List of Values>.) [3 Points]

Phase 1 Reduce would do the following:

```
reduce:
input: <node, {(currdist1), (node1, dist1), (node2, dist2), (node3, dist3)},
      {(currdist2)},
      {(currdist3)}>
output:
mindist = min(currdist1, currdist2, currdist3)
<node, {mindist, (node1, dist1), (node2, dist2), (node3, dist3)}>
```

From the given example,

```
input to Phase 2 reduce:
<A, {3}, {inf, (Sally, 3), (Th1, 1)}>
<B, {5}, {inf, (Th1, 4), (Th2, 6), (Sally, 5), (C,1)}>
...
output to Phase 2 map (next iteration):
<A, {3, (Sally, 3), (Th1, 1)}>
<B, {5, (Th1, 4), (Th2, 6), (Sally, 5), (C,1)}>
...
```

3 DNS (14 Points)

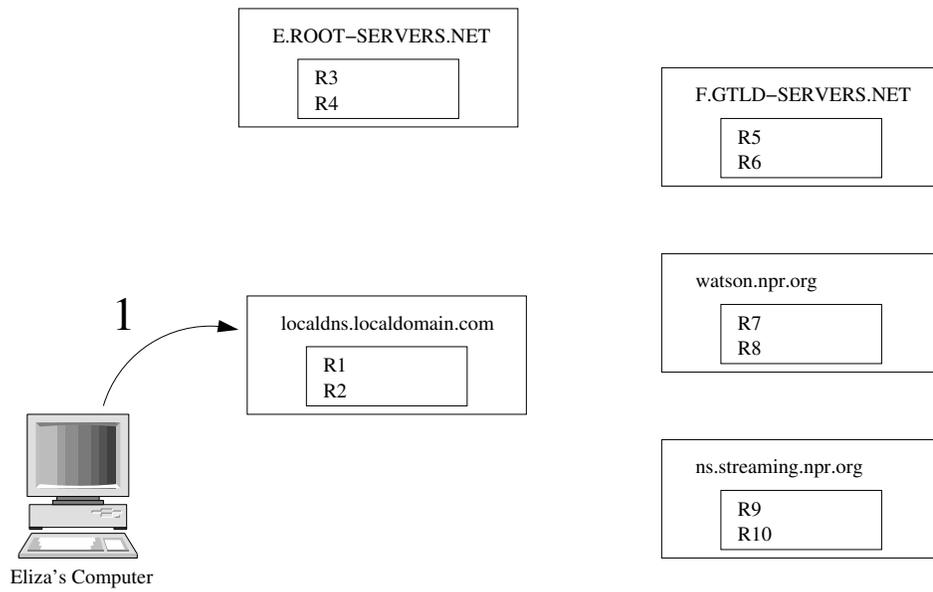
- Elisa wants to listen to the National Public Radio news over the Internet. She starts her favorite audio player and points it to `ra1.streaming.npr.org`. The audio player calls `gethostbyname()` with the given name to obtain the IP address of the server. As a result of the `gethostbyname()` call, the local resolver in Elisa's machine contacts the local DNS server to translate the host name into an IP address. The local DNS server performs an iterative lookup. The table below contains the DNS distributed database. A row corresponds to a DNS record. The records are grouped by DNS server.

Record #	Name	TTL (sec)	IN	Type	Value
localdns.localdomain.com					
R1	.	262542	IN	NS	E.ROOT-SERVERS.NET.
R2	E.ROOT-SERVERS.NET.	348942	IN	A	192.203.230.10
E.ROOT-SERVERS.NET					
R3	org.	172800	IN	NS	F.GTLD-SERVERS.NET
R4	F.GTLD-SERVERS.NET	172800	IN	A	192.35.51.30
F.GTLD-SERVERS.NET					
R5	npr.org	172800	IN	NS	watson.npr.org.
R6	watson.npr.org.	172800	IN	A	205.153.37.175
watson.npr.org					
R7	streaming.npr.org.	172800	IN	NS	ns.streaming.npr.org.
R8	ns.streaming.npr.org	172800	IN	A	205.153.36.175
ns.streaming.npr.org					
R9	audio.streaming.npr.org.	172800	IN	CNAME	ra1.streaming.npr.org.
R10	ra1.streaming.npr.org.	10	IN	A	205.153.36.175

In the figure below, draw arrows to indicate the sequence of queries and responses exchanged among the different machines. Label each arrow with a sequence number, and fill in the table below to indicate the following information:

- Sequence number indicating the ordering of the message exchanges.
- Message Type: use Q for Query or R for Response.
- Data: For queries use the value of the question data. For responses, specify the record ID(s) returned, if any, from the first column in Figure 1, e.g., R1, R2
- You may use abbreviations for host names, e.g. "ra1" rather than `ra1.streaming.npr.org`.

The figure already contains an arrow indicating the first message from the local resolver to the local DNS server. The sequence number is 1 (first message), type = Q (query) and the data is the host name the application wants to resolve (`ra1.streaming.npr.org`). To make your sequence as simple as possible, assume the server includes both the A and NS records when applicable, so include both of them in the corresponding message. **[8 Points]**



Seq	Type	Data
1	Q	ra1.streaming.npr.org(A)

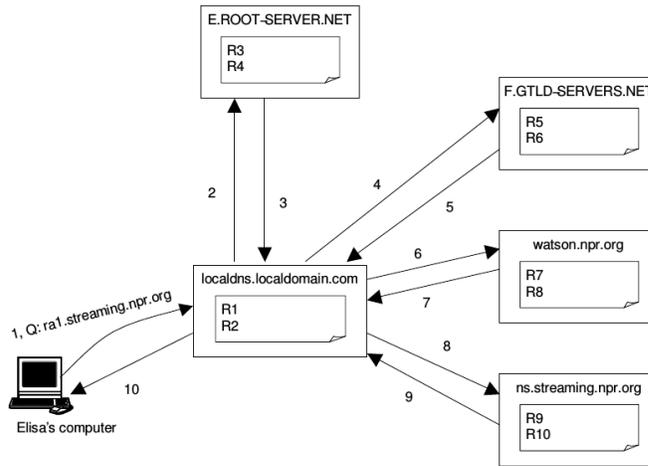


Figure 2: DNS Query & Response sequence.

Seq #	Type	Data
1	Q	ra1.streaming.npr.org (A)
2	Q	ra1.streaming.npr.org (A)
3	R	R3 (NS), R4 (A)
4	Q	ra1.streaming.npr.org (A)
5	R	R5 (NS), R6 (A)
6	Q	ra1.streaming.npr.org (A)
7	R	R7 (NS), R8 (A)
8	Q	ra1.streaming.npr.org (A)
9	R	R10 (A)
10	R	R10 (A)

2. Eliza repeats her query two minutes later. Show what happens for this subsequent query. [6 points]

-2 points per missing message

TTL is only 10 seconds for the last A record...therefore

2 Q ra1(A) (from localdns to ns.streaming.npr.org)

3 R R10(A) (from ns.streaming.npr.org to localdns)

4 R R10(A) (from localdns to Eliza)

4 DIG (18 points)

In this problem, we will use dig tool available on Linux and Mac OS to explore DNS servers. You can read about it using `man dig`. Recall that a DNS server higher in the DNS hierarchy delegates a DNS query to a DNS server lower in the hierarchy, by sending back to the DNS client the name of that lower-level DNS server (assuming no recursion is specified). **For each of the following questions, show the sequence of commands that you ran on your shell with dig.** *Hint:* Be sure to use the `+norecurse` option to dig, and remember that you will need to specify different target DNS

servers (@) each time. Your queries should look like:

```
dig +norecurse @< targetserver> RecordToResolve RecordType.
```

1. Starting with a root DNS server (from one of the root servers [a-m].root-servers.net), initiate a sequence of queries using dig for the A-type record for www.blog.xkcd.com without using recursion. Be sure to also show the list of the names of DNS servers in the entire delegation chain starting from the root in answering your query. **[5 points]**

```
a.root-servers.net
$ dig +norecurse @a.root-servers.net www.blog.xkcd.com
a.gtld-servers.net. 172800 IN A 192.5.6.30
$ dig +norecurse @a.gtld-servers.net www.blog.xkcd.com
dns1.p03.nsonone.net. 172800 IN A 198.51.44.3
$ dig +norecurse @dns1.p03.nsonone.net www.blog.xkcd.com
www.blog.xkcd.com. 3600 IN A 208.118.225.100
```

Depending on the various root servers you chose, your names may differ by a letter or a number, as well as differing IP addresses.

2. Repeat the same procedure as above for www.csd.cs.cmu.edu **[5 points]**

```
a.root-servers.net
$ dig +norecurse @a.root-servers.net www.csd.cs.cmu.edu
a.edu-servers.net. 172800 IN A 192.5.6.30
$ dig +norecurse @a.edu-servers.net www.csd.cs.cmu.edu
nsauth1.net.cmu.edu. 172800 IN A 128.2.1.8
$ dig +norecurse @nsauth1.net.cmu.edu www.csd.cs.cmu.edu
AC-DDNS-1.NET.cs.cmu.edu. 600 IN A 128.2.184.227
$ dig +norecurse @AC-DDNS-1.NET.cs.cmu.edu www.csd.cs.cmu.edu
SCS-MAN-SITES.WEB.CS.CMU.EDU. 3600 IN A 128.2.217.31
```

Depending on the various root servers you chose, your names may differ by a letter or a number, as well as differing IP addresses.

3. Repeat the same procedure for 81.183.132.209.in-addr.arpa. This time, dig for the PTR-type record. **[5 points]**

```
a.root-servers.net
$ dig +norecurse @a.root-servers.net ptr 81.183.132.209.in-addr.arpa
a.in-addr-servers.arpa. 172800 IN A 199.212.0.73
$ dig +norecurse @a.in-addr-servers.arpa ptr 81.183.132.209.in-addr.arpa
209.in-addr.arpa. 86400 IN NS u.arin.net.
$ dig +norecurse @u.arin.net ptr 81.183.132.209.in-addr.arpa
183.132.209.in-addr.arpa. 86400 IN NS ns1.redhat.com.
$ dig +norecurse @ns1.redhat.com ptr 81.183.132.209.in-addr.arpa
81.183.132.209.in-addr.arpa. 600 IN PTR www.redhat.com.
```

4. Use `dig -x IP addr` to perform reverse DNS lookup for the IP address 209.132.183.81. What is the domain name associated with this IP address? What is the type of the DNS record in the answer section? Compare the answer section to part 3's answer. [3 points]

```
$ dig -x 209.132.183.81
81.183.132.209.in-addr.arpa. 600 IN PTR www.redhat.com.
```

5 Web and Peer-to-Peer (14 points)

1. Provide three reasons a company might prefer to pay Akamai to host their webpage instead of putting it onto a peer-to-peer network (such as Napster) for free. [6 points]

- 1.1. It gets a normal DNS name.
- 1.2. Akamai will try very hard to keep it up whereas peer-to-peer systems will only host material while there's a peer that wants to.
- 1.3. It's difficult to ensure the integrity of a file (webpage) in a p2p network.

2. We saw no examples of chunk-based peer-to-peer networks that use flooding. What would make such a network inefficient? [4 points]

For each query, a lot of messages are sent under flooding. Making the network chunk based instead of file based would increase the number of queries leading to even more messages and overhead.

3. While not strictly true, people tend to view hash tables as offering constant time look up in practice. (The possibility of a bad hash function leading to many collisions is why it is not strictly constant time.) Distributed Hash Tables (DHTs), on the other hand, only offer $O(\log n)$ -time lookup where n is the number of nodes in the DHT. This is odd, since the two appear to be equivalent, i.e. simply map each entry in a traditional hash table onto a node in the DHT. What property or requirement of a DHT makes this approach impractical in practice. [4 points]

The number of nodes may change over time, and it is not practical to move the data around everytime that happens.

6 Consistent Hashing (12 points)

David is designing a distributed hash table with n nodes. The table will store values with m -bit keys; each node in the DHT has an ID obtained by hashing the nodes name (e.g., $ID_1 = h(\text{"node1"})$). He is considering two schemes for assigning key-value pairs to nodes responsible for storing them:

- **Scheme 1:** Use consistent hashing. The node responsible for key k is the first node whose ID is equal to or follows k in the identifier space (modulo 2^m).

- **Scheme 2:** Order the nodes numerically by their IDs. If a nodes position in this ranking is r (where $r \in [0, n)$), it is responsible for keys in the range $[r \cdot 2^m, (r + 1) \cdot 2^m)$.

1. What is one advantage of scheme 1? **[3 points]**

Adding in a new node results in less key redistribution.

2. What is one advantage of scheme 2? **[3 points]**

Querying a key will be more efficient since the distribution of keys is more standard. (It's easier to compute node ID when given a key)

3. Consider a new node joining the DHT (for a new total of $n + 1$). On average, for each scheme, what fraction of the key space will be assigned to a different node as a result? For simplicity, assume the following:

- The new nodes ID is higher than any current nodes ID.
- The IDs of the current nodes are evenly distributed: $ID_i = \frac{i}{n} * 2^m$ **[6 points]**

In scheme one, on average the fraction of the key space which will be reassigned will be $2^m / (n + 1) / 2$, since on average the last key's key space will be divided in half. In scheme two, on average half of the key space will be reassigned. (work it out mathematically with small numbers of nodes to see why)

7 Chord and DHT (10 points)

Skype uses a custom protocol to determine whether a user is logged in, where they are located, and what ports they are listening on. Suppose that you have set out to build your own peer-to-peer telephony system.

1. Briefly, how could you use the Chord distributed hash table to maintain a directory of users that was stored entirely on the end-users computers, with no centralized infrastructure? **[4 points]**

Chord could be used to build a dictionary, where a user's name was used as a key, and the value associated with that key included information about the user such as whether they were currently logged on, and if so, where, and on what ports were they listening.

2. Your telephony system catches on very quickly. Soon you have 100,000 users worldwide. However, users start to complain that it takes a long time to perform directory lookups. Assume that the average latency between users is 100ms. Explain why are lookups taking so long and estimate how long they take. **[6 points]**

Chord requires up to $\log n$ messages to complete a lookup, and when n gets to be in the tens of thousands, this is ten to fifteen lookups.

Second, there's no locality in chord, so when the users are located all over the world, each message may have to travel a very long distance.

Third, it might be that users are not staying on the system very long, and there's a lot of overhead (e.g., $\log^2 n$ messages) when someone leaves or joins just to repair pointers in Chord, and on top of this you have to deal with moving data around.