Homework5 -- Storge for Data Intensive Services

Name: Qing Liu

Student ID: 6130116184

Class: 164 of Computer Sciense and Technology

Questions

1. HDFS is implemented as a user-level file system vs an in-kernel file-system. (a) What is the advantage of this in the context of Hadoop?
2. The output of a Mapper is written into the local filesystem instead of the global filesystem. Why? Your answer should explain both why writing into the global file system would be undesirable as well as why it would be of minimal benefit.
3. Why does Hadoop sort records en route to a Reducer? How would it affect things if these records were processed by the Reducer in the order in which they were received from the various Mappers?
4. How is the failure of a Mapper or Reduce managed?
5. In a typical Map-Reduce graph algorithm, what data structure is used to represent the graph? Why?
6. In a typical Map-Reduce graph algorithm, how many Map-Reduce phases are typically necessary before the graph can be traversed? Why?

Answers

A1. According to the assumptions and goals of HDFS, here are the advantages:

- HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system’s data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS. If it was a kernel file system, it has to deal with hardware, which greatly increases the difficulty of implementation.
- HDFS is designed more for batch processing rather than interactive use by users. The emphasis is on high throughput of data access rather than low latency of data access. Applications that run on HDFS need streaming access to their data sets. By this way, it can improve data throughput.

- HDFS is designed to facilitate porting from one platform to another. This helps to widely adopt HDFS as the preferred platform for a large number of applications. Cross-platform can only be achieved through application or level.

A2.

- It does not use valuable cluster bandwidth. This is called data location optimization. Sometimes, however, all nodes that carry copies of HDFS blocks for mapping task input splitting are running other mapping tasks, so the job scheduler will look for free mapping slots on nodes with one of the blocks in the same rack. Sometimes even this is impossible, so use a node off the rack, which leads to network transmission between racks.

- Don't need the fault tolerance of HDFS. If the job dies halfway, we can always just re-run the map task.

A3.

- A MapReduce job consists of Map phase and Reduce phase, which sort data. In this sense, the MapReduce framework is essentially a Distributed Sort. In Map stage, Map Task will output a file sorted by key (using fast sort) on the local disk (multiple files may be generated in the middle, but will eventually be merged into one). In Reduce stage, each Reduce Task will sort the received data, so that the data will be divided into several groups according to Key, and then handed over to reduce in groups. In fact, the sorting in Map stage is to reduce the sorting load on Reduce side.

- It's not that sort is good for subsequent operations, but that sort brings many benefits to many applications and subsequent application development. Imagine that distributed computing frameworks don't provide sorting for you. Why do we use it?

A4.

- The most common occurrence of this failure is when user code in the map or reduce task throws a runtime exception. If this happens, the task JVM reports the error back to its parent application master before it exits. The error ultimately makes it into the user logs. The application master marks the task attempt as failed, and frees up the container so its resources are available for another task.

- Another failure mode is the sudden exit of the task JVM perhaps there is a JVM bug that causes the JVM to exit for a particular set of circumstances exposed by the MapReduce user code. In this case, the node manager notices that the process has exited and informs the application master so it can mark the attempt as failed.

- Hanging tasks are dealt with differently. The application master notices that it hasn’t received a progress update for a while and proceeds to mark the task as failed. The task JVM process will be killed automatically after this period. The timeout period after which tasks are considered failed is normally 10 minutes and can be configured on a per-job basis (or a cluster basis) by setting the mapreduce.task.timeout property to a value in milliseconds.

A5.

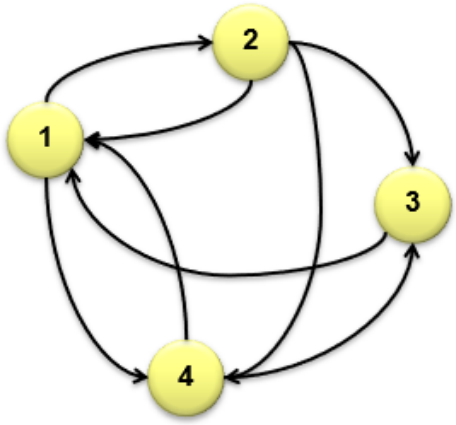
- Graph algorithm typically involve in performing computations at each node: based on node features, edge features, and local link structure and propagating computations: “traversing” the graph. We can use `G=(V,E)` to represent the graph, where V represents the set of vertices (nodes) and E represents the set of edges (links). Both vertices and edges may contain additional information. So, there are two common representations for graph: **Adjacency matrix** and **Adjacency list**.

  - **Adjacency Matrices**

Represent a graph as an *n* x *n* square matrix *M*

- *n* = |V|
- M_{ij} = 1 means a link from node *i* to *j*

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0

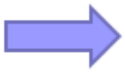


    - *Advantages*:
      - Amenable to mathematical manipulation
      - Iteration over rows and columns corresponds to computations on outlinks and inlinks
    - *Disadvantages*:
      - Lots of zeros for sparse matrices
      - Lots of wasted space

  - **Adjacency Lists**

Take adjacency matrices… and throw away all the zeros

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	1	0	0	0
4	1	0	1	0



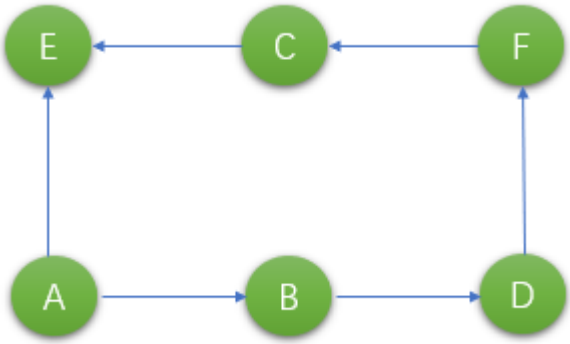
1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

- *Advantages*
  - Much more compact representation
  - Easy to compute over outlinks
- *Disadvantages*
  - Much more difficult to compute over inlinks

A6.

I think there are 4 phases:

Suppose we have this graph:



- Firstly, this data would likely to be represented in our Hadoop cluster as list of connections, like:

Node	Connections	Distance
A	B,E	0

Node	Connections	Distance
B	C,D	x
C	E	x
D	F	x
E		x
F	C	x

The only distance we are sure of in the beginning is that the distance between ‘A’ and ‘A’ is zero. All other points we’ll put as x, for unknown. So in the first pass, we feed the table into our mappers and we produce a key-value pair for each node and connection.

- Secondly, when a mapper reads in the line for Node 'A', it will emit three pairs:

Node	Connections	Distance
A	B,E	0

becomes

A	B, E	0
B	_	1
E	_	1

The first effectively reiterates what is already known about ‘A.’ The second two are new information about ‘B’ and ‘E.’ Since they are connected to ‘A,’ their distance becomes the distance to ‘A’ plus 1, which makes them 1 (0 + 1 = 1). That single line we’re parsing does not contain any information about what ‘B’ and ‘E’ are connected to, so we leave those fields empty.

- Thirdly, Mapper Output

  Once the mappers process all the lines, they’ll produce an output like this (but not necessarily in this order):

A	B, E	0
B	_	1
E	_	1
B	C, D	x
C	_	x
D	_	x
C	E	x
E	_	x
D	F	x
F	_	x
E		x
F	C	x
C	_	x

Whenever we process a node we don't have the distance to, we won't know the distance to any of its connected nodes either, so we'll leave them blank for now.

- Fourthly, we separate out that data by node and send it to the reducers. The reducers will strip out all redundant information and reduce all values for a node back down to a single entry using the smallest known value for "distance" known. So the node that processes 'B' will get:

B	_	1
B	C, D	x

which will then be reduced to:

B	C, D	1

Reducer Output:

Node	Connections	Distance
A	B,E	0
B	C,D	1
C	E	x

Node	Connections	Distance
D	F	x
E		1
F	C	x

We have now completed a single iteration. We know the distance for all nodes within one jump from ‘A.’ Not too impressive, but every iteration we do will increase our knowledge to the next depth of the tree. We can keep doing this up until the graph stops changing.