CS546 "Parallel and Distributed Processing"

Homework 7

1. **(10 points)** Why was Hadoop platform for Big Data processing introduced and became so popular? What features made it such a widely-adopted solution for data processing? What are its benefits over existing solutions? Explain briefly.

Ans: The traditional data analysis involved complex processing like regression, etc upon small data sets, usually representative samples of larger population of data. Computations of this type are dependent on the size and performance of the processor and main memory. To improve the computation speed or the amount of data a computer is able to process, faster processors and more ram is required. To solve this Distributed system could be used where we multiple systems could be used to perform single job. But with this programming complexities increased, bandwidth constraints and challenge to support partial failures. The core Hadoop concepts were as follows

a. applications are written in high level languages
b. nodes talk to each other as little as possible
c. data is distributed in advanced
d. data is replicated for increased availability and reliability
e. hadoop is scalable and fault tolerant

Features that made Hadoop Popular

a. **Ability to store and process huge amounts of any kind of data, quickly**. With data volumes and varieties constantly increasing, especially from social media and the Internet of Things (IoT), that's a key consideration.
b. **Computing power**. Hadoop's distributed computing model processes big data fast. The more computing nodes you use, the more processing power you have.
c. **Fault tolerance**. Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.
d. **Flexibility**. Unlike traditional relational databases, you don't have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.
e. **Low cost**. The open-source framework is free and uses commodity hardware to store large quantities of data.
f. **Scalability**. You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

2. **(10 points)** What is MapReduce and what are its characteristics? List its advantages and disadvantages. Compare it with MPI-based approaches to parallelize a certain task.

Ans: MapReduce is a processing technique and a program model for distributed computing based on java.

The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines

those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.
The advantages of MapReduce programming are,

a. **Scalability** Hadoop MapReduce programming enables business organizations to run applications from a huge number of nodes that could involve the usage of thousands of terabytes of data.
b. **Flexibility** Business organizations can make use of Hadoop MapReduce programming to have access to various new sources of data and also operate on different types of data, whether they are structured or unstructured.
c. **Security and Authentication** MapReduce works with HDFS and HBase security that allows only approved users to operate on data stored in the system.
d. **Parallel processing** MapReduce divides tasks in a manner that allows their execution in parallel**.**

The disadvantages of MapReduce are,

a. MapReduce is not suitable for Real-Time processing, example When you need to handle streaming data. MapReduce is best suited to batch process huge amounts of data which you already have with you.
b. It is not always intuitive to implement everything as a MapReduce program.
c. It is not good if there is inter process communication

In the following table For various task which is preferable has been shown

Sno	Task	MPI	MapReduce
1	programming efficiency is an issue, i.e., how fast can I have my application up and running, and in a scalable fashion	No	Yes
2	Fault Tolerance	No	Yes
3	exploit data locality	Yes	No

3. **(20 points)** Provide pseudocode for sorting integers in an out-of-core fashion (i.e., integers cannot fit in memory) both in MapReduce (both mapper and reducer code) and in MPI. Think about the flow of sorting: reading the input, perform the sorting algorithm, merge all intermediate results, write the final sorted output.

Ans: MPI:

```
get input in A
if(root)
        partitionData and broadcast
if not root
        receive data
sortData
send sorted data to root
if root
        collect and merge data
```

MapReduce:

```
Mapper
        Map data as key,value where key is weight and value is value
Reduce
        Then data is shuffled for increasing order of weights
```

4. **(10 points)** Provide a table of features for Distributed File Systems (HDFS) and Parallel File Systems (PVFS). Discuss briefly similarities and differences.

Features	PVFS	HDFS
Hierarchical storage management	Allows sufficient usage of disk drives with different performance characteristics	
High performance support for MapReduce applications	Stripes data across disks by using metablocks, which allows a MapReduce split to be spread over local disks	Places a MapReduce split on one local disk
High performance support for traditional applications	• Manages metadata by using the local node when possible rather than reading metadata into memory unnecessarily • Caches data on the client side to increase throughput of random reads • Supports concurrent reads and writes by multiple programs	

Features	PVFS	HDFS
	• Provides sequential access that enables fast sorts, improving performance for query languages such as Pig and Jaql	
High availability	Has no single point of failure because the architecture supports the following attributes: • Distributed metadata • Replication of both metadata and data • Node quorums • Automatic distributed node failure recovery and reassignment	Has a single point of failure on the NameNode, which requires it to run in a separate high availability environment
POSIX compliance	Is fully POSIX compliant, which provides the following benefits: • Support for a wide range of traditional applications • Support for UNIX utilities, that enable file copying by using FTP or SCP • Updating and deleting data • No limitations or performance issues when using a Lucene text index	Is not POSIX compliant, which creates the following limitations: • Limited support of traditional applications • No support of UNIX utilities, which requires using the hadoop dfs get command or the put command to copy data • After the initial load, data is read-only • Lucene text indexes must be built on the local file system or NFS because updating, inserting, or deleting entries is not supported
Ease of deployment	Supports a single cluster for analytics and databases	Requires separate clusters for analytics and databases
Other enterprise level file system features	• Workload isolation • Security	
Data replication	Provides cluster-to-cluster replication over a wide area network	HDFS Snapshots on a subtree or on your entire file system to create read-only copies of the file system. These copies can

Features	PVFS	HDFS
		be used for data backup and recovery.

5. **(10 points)** How does data distribution works on HDFS? Who is responsible for distributing data? What would you optimize in the distribution policies to make the system faster and more reliable?

Ans: On HDFS the data is distributed across nodes. The information of what data is where, metadata, is stored in the NameNode. Essentially, the data is distributed based on locality of produced and consumers of data. Also the data is duplicated primarily for fault tolerance, but this also is used to get better locality for the request. For example, generally data is duplicated within same rack and different rack. When a request comes the nameNodes searches for the closes node which has the data and gives client that address.

The NameNode is responsible for policies and distribution of data across dataNodes.

To make the system faster and reliable the policies needs to be optimized to improve locatity based on what consumers of that data. Also replication policy should be based on the hardware's status. That is a hardware which has more chances of failing should have more replications than one which is more fault tolerant.

6. **(10 points)** Discuss how the fault tolerance features in HDFS work? Compare it with hardware based (RAID) approaches. What other approaches for fault tolerance are out there (think erasure coding) ?

Ans: HDFS achieves fault tolerance mechanism by replication process. In HDFS whenever a file is stored by the user, then firstly that file is divided into blocks and then these blocks of data are distributed across different machines present in HDFS cluster. After this, replica of each block is created on other machines present in the cluster. By default, HDFS creates 3 copies of a file on other machines present in the cluster. So, due some reason if any machine on the HDFS goes down or fails, then also user can easily access that data from other machines in the cluster in which replica of file is present. Hence HDFS provides faster file read and write mechanism, due to its unique feature of distributed storage. For eg: Suppose there is a user data named FILE. This data FILE is divided in into blocks say B1, B2, B3 and send to Master. Now master sends these blocks to the slaves say S1, S2, and S3. Now slaves create replica of these blocks to the other slaves present in the cluster say S4, S5 and S6. Hence multiple copies of blocks are created on slaves. Say S1 contains B1 and B2, S2 contains B2 and B3, S3 contains B3 and B1, S4 contains B2 and B3, S5 contains B3 and B1, S6 contains B1 and B2. Now if due to some reasons slave S4 gets crashed. Hence data present in S4 was B2 and B3 become unavailable. But we don't have to worry because we can get the blocks B2 and B3 from other slave S2. Hence in unfavourable conditions also our data doesn't get lost. Hence HDFS is highly fault tolerant.

RAID storage solutions have different levels – most commonly used are:

- RAID 0 – provides no fault tolerance, but it increases disk speed 2x or better.
- RAID 1 – mirrors the data on multiple disks to provide fault tolerance, but requires more space for less data.
- RAID 5 – strips the disks similar to RAID 0, but doesn't provide the same amount of disk speed. Has fault tolerance without the loss of any data.

- RAID 6 – minimum of four disks. Same as RAID 5, but the system can fail twice and not lose any data.
- RAID 7.3 – This new RAID option answers the need for a triple-parity RAID. With RAID 5 and RAID 6 beginning to become inadequate, this option is beginning to take the steps necessary to provide a more reliable storage option that RAID 5 and RAID 6.
- RAID 10 – this option is costly, but it combines RAID 0 and RAID 1. The RAID 10 option requires four disks, and can continue to operate without loss of any data so long as the failures occur in different subgroups.

The techniques commonly used for designing fault tolerance

Checkpoint and rollback recovery

All general-purpose fault tolerance technique rely on the same idea: introduce automatically computed redundant information, and use this redundancy to mask the occurrence of failures to the higher level application

a. Process checkpointing : The goal of process checkpointing is to save the current state of a process. In current HPC applications, a process consists of many user-level or system-level threads, making it a parallel application by itself. Process checkpointing techniques generally use a coarse-grain locking mechanism to interrupt momentarily the execution of all the threads of the process, giving them a global view of its current state, and reducing the problem of saving the process state to a sequential problem.
b. Coordinated checkpointing: Distributed checkpointing protocols use process checkpointing and message passing to design rollback recovery procedures at the parallel application level.

Probabilistic models for advanced methods

a. Fault prediction: A fault predictor is a mechanism that warns the user about upcoming faults on the platform.
b. Replication: Replication consists in duplicating all computations.

7. **(10 points)** Provide a description for the ideal workloads for HDFS. What about the worst-case scenario?

Ans: Batch Processing Systems are ideal workload for HDFS. Batch processing is the execution of a series of jobs in a program on a computer without manual intervention (non-interactive). Strictly speaking, it is a processing mode: the execution of a series of programs each on a set or "batch" of inputs, rather than a single input (which would instead be a custom job). However, this distinction has largely been lost, and the series of steps in a batch process are often called a "job" or "batch job". These require high throughputs for sequential reads and writes.

The worst workload for HDFS is Real time processing applications. , example When you need to handle streaming data. HDFS is best suited to batch process huge amounts of data which you already have with you. Having stream of data coming in puts too much overhead on the system. In HDFS the file exists only as a directory entry, it shows as having zero length until the file is closed. This means if

data is written to a file for an extended period without closing it, a network disconnect with the client will leave you with nothing but an empty file for all your efforts. This may lead you to the conclusion that it would be wise to write small files so you can close them as soon as possible.

The problem is Hadoop doesn't like lots of tiny files. Since the HDFS metadata is kept in memory on the NameNode, the more files you create, the more RAM you'll need to use. From a MapReduce prospective, tiny files lead to poor efficiency. Usually, each mapper is assigned a single block of a file as input (unless you have used certain compression codecs). If you have lots of tiny files, the cost of starting the worker processes can be disproportionally high compared to the data it is processing. This kind of block fragmentation also results in more mapper tasks increasing the overall job run times.

8. **(20 points)** What storage needs led to the birth of NoSQL schema? Why did object storage become the de-facto solution in web services? Discuss the weaknesses and the strengths of a key-value store.

Ans: the reason for the introduction of NoSQL is because RDMBS cannot actually cope with some modern application because it cannot meet the requirement of these applications. This is because RDBMS have the following problem:

a. Low throughput: Most RDBMS cannot handle large volume of data store
b. RDBMS are built to scale vertically: vertical scaling means adding more hardware to the existing machine (Petter 2013), whereas horizontal scaling means adding more machine to a cluster.
c. Object –Relational mapping is expensive: it takes a lot of time to map the applications object model and the databases relational model
d. The thinking "one size fits all" is flawed: RDBMS design is seen as a general tool that can meet the entire requirement that an application can have on data management, this is not possible
e. ACID (Atomicity, Consistency, Isolation and Durability) is not always needed: with relational database system ACID is a requirement and it is a trade off for performance. But not all application requires ACID which leads to loss of performance for nothing.

Advantages of Key-Value store

a. **Concurrency** – In Key/Value Store, concurrency is only applicable on a single key, and it is usually offered as either optimistic writes or as eventually consistent. In highly scalable systems, optimistic writes are often not possible, because of the cost of verifying that the value haven't changed (assuming the value may have replicated to other machines), there for, we usually see either a key master (one machine own a key) or the eventual consistency model.
b. **Scaling Up** – In Key Value stores, there are two major options for scaling, the simplest one would be to shard the entire key space. That means that keys starting in A go to one server, while keys starting with B go to another server. In this system, a key is only stored on a single server. That drastically simplify things like transactions guarantees, but it expose the system for data loss if a single server goes down. At this point, we introduce replication.
c. **Usages** – Key Value stores shine when you need to access the data by key

Disadvantages

a. **Queries** – there really isn’t any way to perform a query in a key value store, except by the key. Even range queries on the key are usually not possible.
b. **Design:** many data structures (objects) can't be easily modeled as key value pairs