1.! (10 points) Provide an overview of PFS. Briefly discuss its design goals, its strengths and weaknesses.

PFS = Parallel File Systems. At first it is a File System, its design goals are to handle concurrent access from multiple clients and handle large datasets that cannot fit in memory in a high-performance way.

The strengths are:

Since PFS provided global shared namespace for files and dictionaries,

it provides **scalability,** which can handle thousands of clients, and the **capability** to distribute large files across multiple nodes. In a cluster environment, large files are shared across multiple nodes, making a parallel file system well suited for I/O subsystems. Thus, it also provides lower management costs for huge capability.

The Weaknesses are:

Since PFS has to sharing metadata across servers, thus there is high demand on network bandwidth.

Since files are separated in several nodes, there's a protentional for data lose in case of server node failure.

2.! (10 points) Compare the file-per-process and shared-file access patterns. Pros and cons.

Shared-file(N-to-1):

Pros:

Increases usability: only one file to keep on by application

Cons:

Can create lock contention and hinder performance on some systems in case of multiple write access to same region(same chunk of a file).

File-per-process(N-to-N):

Pros:

Avoids lock contention on file systems that use locks to maintain POSIX consistency.

Cons:

Impossible to restart application with different number of tasks

3.! (10 points) Describe how data distribution works in PFS.

Parallel file systems distribute data across multiple storage nodes. That is:
Receive file from Client computing node, update file metadata, then it was sharing metadata with server nodes, thus the global namespace is updated so all nodes can see it, then the files are distributed to different Server Nodes.

4.! (10 points) Locking in PFS. Discuss and list some of the challenges.

Complexity. Since all files, in PFS, usually broken into lock units, which means system needs to implement distributed lock manager to allow possible concurrently writing. This increase the complexity of whole system implementation.

Failure.

Let's say a lock unit is acquired by client A, and Client A eventually failure, then the file unit is locked and no any client can acquire this lock because it is not released anymore. This can be solved by implement a lock timeout or heartbeat tracking. When a client is disconnected, then the lock is automatically released by lock manager.

In other hand, if a file unit failure, the locks are missing. This may result client failure as well. Especially when the failure is occurred during I/O. The lock queue can also be broken because of this. An overall solution is that simply avoid to write overlapping region concurrently.

Performance.

Locks are expensive, there are Round-trip latencies between clients and DLM.

5.! (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.

At first it is **Open source** software framework designed for storage and processing of large scale data on clusters of commodity hardware. Which means it is free, and supported by large open source community. The reason why was Hadoop introduced is that nowadays we need a solution that process huge datasets on large clusters of computers and must be **efficient**, **reliable**, **easy to use**. Hadoop platform indeed satisfied those requirements compare to other solutions so it is so popular.

The core design goal of Hadoop was to distribute the data as it is initially stored. Which means each node can then perform computation on the data it stores without moving the data for the initial processing. Also, **Hadoop provide handful API**, made applications can be written in a high-level programming language. Which provided an easy-to-use way. Additionally, Nodes communicate are reduced to as little as possible (share-nothing architecture). Data is spread among the machines in advance to provide transparent file distribution. In conclusion, Hadoop is reliable, high-performance, cost efficient, and free.

Compare to other solutions, Hadoop has much benefits not only it is open source but also it provides several components that can satisfy nearly all the needs.

6.! (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.

MapReduce is a method for distributing a task across multiple nodes. Each node processes data stored on that node.

MapReduce Consists of two phases:

Map: (K1, V1) -> (K2, V2)

Reduce: (K2, list(V2)) -> list(K3, V3)

Advantages:

Automatic parallelization and distribution. Fault tolerance (individual tasks can be retried)
A clean abstraction for developers. Also, Hadoop as most popular solution comes with standard status and monitoring tools for MapReduce.

7.! (10 points) Provide a table of features for Distributed File Systems (HDFS) and Parallel File Systems (PVFS). Discuss briefly similarities and differences.

	HDFS	PVFS
Deployment model	Co-locates compute and storage on the same node (beneficial to Hadoop/MapReduce model where computation is moved closer to the data)	Separate compute and storage nodes (easy manageability and incremental growth)
Concurrent writes	Not supported – allows only one writer per file	Guarantees POSIX sequential consistency for non-conflicting writes, i.e., optimized writes to different regions of a file
Small file operations	Not optimized for small files; client-side buffering aggregates many small requests to one file into one large request	Uses few optimizations for packing small files, but the lack of client-side buffering or caching may result in high I/O overhead for small write requests
Append mode	Write once semantics that allows file appends using a single writer only	Full write anywhere and rewrite support
Buffering	Client-side readahead and write-behind staging improves bandwidth, but reduces durability guarantees and limits support for consistency semantics	No client-side prefetching or caching provides improved durability and consistency for concurrent writers
Data layout	Exposes mapping of chunks to data-nodes to Hadoop applications	Maintains stripe layout information as extended attributes but not exposed to applications
Fault tolerance	Uses rack-aware replication with, typically, three copies of each file	No file system level support; relies on RAID subsystems attached to data servers
Compatibility	Custom API and semantics for specific users	UNIX FS API with most POSIX semantics

Both HDFS and PVFS divide a file into multiple pieces, called chunks in HDFS and stripe units in PVFS, that are stored on different data servers. Each HDFS chunk is stored as a file in the lower-layer local file system on the data server. Unlike HDFS's file-per-chunk design, PVFS splits a file into an object per data server, where each object includes all stripe units of that file stored on that data server as a single file in the underlying local file system. The chunk size and stripe unit size are configurable parameters; by default, HDFS uses 64 MB chunks and PVFS uses 64 KB stripe units, although modern PVFS deployments use larger stripe units (2-4 MB) to get higher data transfer bandwidth.

All metadata operations may be handled by a single server in HDFS, but a cluster will typically configure multiple servers as primary-backup failover pairs, and in PVFS, metadata may be distributed across many active MDSs as well. All file data is stored on persistent storage on a different set of servers called I/O servers in PVFS and data servers in HDFS.

8.! (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?

HDFS operates on top of an existing filesystem Files are stored as 'Blocks', each Block is replicated across several Data Nodes. NameNode stores metadata and manages access to data and metadata. The Client retrieve/update metadata via communication to NameNode, and writes to replicated data nodes. NameNode is responsible for distributing data.

To make it faster, we can do some research on replica selection, such as nearest preferred reading to reduce network latency, also we can implement some priority based replica placement like thread priority. Like, for critical priority task, we distribute it first, although the total time-consuming stay the same but it can accelerate whole system's speed. For more reliable, we could make trade-offs, like for critical task, we replicate it more, for low priority task, we replicate it less.

9.! (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)?

HDFS handles faults by the process of replica creation. The replica of data is created on different machines in the HDFS cluster. So whenever if any machine in the cluster goes down, then data can be accessed from other machine in which same copy of data was created. HDFS also maintains the replication factor by creating replica of data on other available machines in the cluster if suddenly one machine fails.

For RAID, Data is replicated or stored using parity bits on multiple disks. If a disk fails, it can be recovered from another replica or recomputed using the parity bits. HDFS replicate data to racks but no disks so HDFS is slower, but HDFS is cheaper since it does not require special hardware configuration.

Erasure coding is a method of data protection in which data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media. However, this method requires more CPU usage and may result more latency.

10.!(10 points) Provide a description for the ideal workload for HDFS. What about the worst-case scenario?

HDFS is best to handle large data processing. That is a skilled user follow design document and divide computing tasks in a right order suggested by apache Hadoop. In general, when PFS is not require, like massive concurrent writing, the HDFS is better. The worst case scenario is that when a application on top of HDFS writes to overlapped region massively, this would hit the weak point of HDFS and have big problem in performance.

11.!(Optional) 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.