**Q1.**

**Minor file issue**: Hadoop was designed to handle massive datasets, not a lot of files that were significantly smaller than the 128 MB default size. The NameNode must hold metadata, such as names, locations, and permissions, for each data unit. It is obvious that millions of tiny files will take up too much memory in the Master Nodes, generating a large number of tasks that will impede processing speed.

**High access latency to the data**. Large data batches can be sent by the system thanks to Hadoop's high throughput. However, latency—the interval of time between a user's activity and a system response—is sacrificed in the process. To put it another way, it will take a while to locate and obtain a single record. Hadoop is not suited for activities requiring almost real-time data access because to its high latency.

**No processing of data in real time**: Only batch processing can be done using MapReduce; real-time analytics tasks and data that is time-sensitive cannot be accommodated.

**Complex environment for programming**: To fully utilise Hadoop, data engineers who have only dealt with relational database management systems and SQL queries in the past must receive training. To dive further into Hadoop coding and efficiently utilise features made available by Java APIs, they must be proficient in Java. It's also critical to comprehend Hadoop's underlying ideas.

**Q2.**

**Expensive hardware**: Because RAM is more expensive than the hard drives that MapReduce uses, Spark operations are more costly.

**Processing that is almost real-time but not quite**: Spark Streaming and in-memory caching facilitate rapid data analysis. However, because the module uses micro-batches, which are tiny collections of events gathered over a set period of time, it won't be fully real-time. Data streams are processed as soon as they are created by true real-time processing technologies.

Spark doesn't work well with IoT solutions because of this. The Apache portfolio contains more advanced real-time analytics solutions. For instance, Apache Flink was created especially to handle real-time data processing. Real-time streams are likewise better handled by Apache Storm over HBase than by Spark.

**Problems with small files**: Similar to Hadoop, Spark struggles to handle a big quantity of little datasets. Processing can be significantly slowed down when there are more files in a batch since there is more metadata to interpret and work to plan.

**Q3.**

In comparison to Hadoop 1.0, Hadoop 2.0 brought forth a number of noteworthy adjustments and enhancements that fixed certain issues and increased the framework's adaptability and scalability. The main distinctions between Hadoop 1.0 and Hadoop 2.0 are as follows:

YARN (Yet Another Resource Negotiator) was introduced with Hadoop 2.0, taking the role of the previous job management system that was MapReduce-centric. A Hadoop cluster may execute numerous data processing engines thanks to YARN, a cluster resource management framework. With this modification, Hadoop became more flexible by supporting workloads other than MapReduce.

Better Resource Management: Hadoop's resource management was greatly improved with YARN. The ResourceManager (RM) and NodeManager (NM) were introduced by YARN. While the NodeManager is in charge of keeping an eye on resource utilisation on specific cluster nodes, the ResourceManager serves as a global task scheduler and resource arbitrator. This enhanced how resources were allocated and used across different apps.

Additional Workload Support: Hadoop 2.0 increased the number of workloads that Hadoop could manage. While batch processing was the main focus of Hadoop 1.0, real-time processing, interactive querying, and other workloads might be accommodated by Hadoop 2.0. It made it possible for several data processing engines to live together on the same cluster.

Enhanced Scalability: Enhanced scalability was a primary design goal for Hadoop 2.0. Hadoop clusters can now accommodate more nodes and manage higher volumes of data thanks to the advent of YARN and other architectural modifications, which makes it appropriate for enterprises with expanding data needs.

Improved Performance: The goal of Hadoop 2.0 was to outperform Hadoop 1.0 in terms of performance. Processing efficiency was increased by separating resource management from job execution using YARN. Hadoop was able to perform a wider variety of workloads more quickly thanks to it.

Hadoop 2.0 was created to be backwards compatible with Hadoop 1.0 so that applications and MapReduce tasks that were already in place may continue to operate on the updated architecture. For enterprises running Hadoop 1.0, this facilitated the switch to Hadoop 2.0.

**Q4.**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**References and Citations:**

Ketu, S., Kumar Mishra, P., & Agarwal, S. (2020, June 30). Performance Analysis of Distributed Computing Frameworks for Big Data Analytics: Hadoop Vs Spark. Computación Y Sistemas, 24(2). https://doi.org/10.13053/cys-24-2-3401

In-Text Citation: (Ketu et al., 2020)

 Pelucchi, M., Psaila, G., & Toccu, M. (2018, December 17). Hadoop vs. Spark: Impact on Performance of the Hammer Query Engine for Open Data Corpora. Algorithms, 11(12), 209. https://doi.org/10.3390/a11120209

In-Text Citation: (Pelucchi et al., 2018)