Concluding MapReduce

Let's conclude the MapReduce system's design.

We'll cover the following

Main highlights

Some of the main highlights of this `MapReduce` system's design include:

1. MapReduce is a simplified and less-complicated model, even for programmers without experience. It's easy to parallelize and distribute computations among workers and achieve fault tolerance. It achieves this by hiding the details of data distribution, parallelization, fault tolerance, locality optimization, and load balancing from the programmer. The MapReduce model is applicable to many use cases.

Point to ponder

Question

How does the `MapReduce` library compare to other parallel processing frameworks, such as Message Passing Interface (MPI)?

Hide Answer

Similar systems like `MapReduce` provide abstractions to the programmer to achieve parallelization, such as Bulk Synchronous Programming and

Message Passing Interface (MPI).

The major difference is that the programmer still needs to have a fairly good understanding of parallelization and needs to manage faults themselves. On the other hand, MapReduce shields programmers from the messy details of distributed and parallel computing and enables them to focus on their business logic.

2. We designed our system to minimize the data sent across the network by reading from local disks and writing intermediate data to the local disks as well. Implementing the locality optimization helped us save network bandwidth, which is a scarce resource.
3. We introduced re-execution to handle stragglers. It helped us counter the issues related to machine failures, intermediate data loss, and slow workers.
4. We can model a wide range of problems as MapReduce computations. For example, Google uses the `MapReduce` library to generate its web search service, sorting, data mining, machine learning, and many more systems.
5. The MapReduce model implementation can scale to large clusters comprising thousands of commodity machines. It efficiently uses these resources and is suitable for many significant computational problems.

In conclusion, MapReduce provides a simpler abstraction for parallel data processing where it manages the intricacies of distributed and parallel computing. The MapReduce system's design is simple and robust against failures on commodity servers. MapReduce relies on local disk persistence and GFS replication to logically checkpoint work done by the `Map` or `Reduce` phases. In the real world, we often concatenate many MapReduce jobs to get the work done. Having persistent checkpoints makes it simpler to move from one stage of a huge MapReduce job to the next. MapReduce has proved to be a breath of fresh air in the era of many-core computing, which expects programmers to be responsible for their program's speed-up by explicit parallelization. It has enabled programmers to benefit from a collection of servers in a straightforward manner.

Back

MapReduce: Evaluation

Next

Quiz on MapReduce

Mark as Completed