2.1 Introduction, Assumptions and Goals

Introduction

HDFS is a distributed file system designed to run on commodity hardware. Although it has many similarities with existing distributed file systems, the differences are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.

Assumptions and goals

1. **Hardware Failure** 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, detections of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

2. **Streaming Data access** Applications that run on HDFS need streaming access to their data sets. They are not general purpose applications that typically run on general purpose file systems. ***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. POSIX imposes many hard requirements that are not needed for applications that are targeted for HDFS. POSIX semantics in a few key areas has been traded to increase data throughput rates.

3. **Large data sets** Applications that run on HDFS have large data sets. A typical file in HDFS is gigabytes to terabytes in size. Thus, HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster. It should support tens of millions of files in a single instance.

4. **Simple coherency model** HDFS applications usually require a write-once-read-many access model for files. A file once created, written and closed need not be changed except for appends and truncates. Appending the content to the end of the files is supported but cannot be updated at an arbitrary point. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model.

5. **"Moving computation is cheaper than moving data"** A computa-

tion requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system; the assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

6. **Portability across Heterogeneous Hardware and software platforms** HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

Sources

1. Main

Requires Further Research

1. POSIX requirements (Streaming data access, Assumptions and goals)