1 Background

The Hadoop Distributed File System (HDFS) consists of processes of different roles running on different nodes. The roles in HDFS involve NameNode, DataNode, JournalNode, Client, and so on. In this paper, we call them components in HDFS. To tolerate failures, a component usually has multiple replications. For example, the HDFS in our tests is composed of 2 NameNode processes, 4 DataNodes, 3 JournalNodes, and 1 Client. We call them the instances of a component.

2 Definition

2.1 Informal Definition

Before we give an informal definition about *Reconfigurability*, we need to discuss how system administrators reconfigure systems in real world. Essentially, there are two methods.

First, system administrators can shutdown the whole system, make whatever reconfiguration needed, and then restart the whole system. Although this method provides a safest way to reconfigure the system, it hurts system availability during the reconfiguration period. We call this kind of reconfiguration offline reconfiguration.

As opposed to offline reconfiguration, system administrators might also be able to make reconfiguration in an online fashion. For example, some systems, such as HDFS, provide native support for online reconfiguration for certain parameters. In addition, for many distributed systems, because components are replicated for fault tolerance, it naturally provides the opportunity to incrementally reconfigure part of components with offline reconfiguration but still keep the whole system online. We call this kind of reconfiguration online reconfiguration.

Intuitively, for any parameter reconfiguration, if it can be performed with online reconfiguration, it can be reconfigured with offline reconfiguration as well. However, the reserve will not always be true. Thus, the informal definition of *Reconfiguration Errors* could be:

For a given parameter *p*, if it can be reconfigured offline but cannot be reconfigured online, we say that parameter *p* is *not online reconfigurable*.

2.2 Interpretation of Informal Definition

Now, we will derive from this informal definition into a formal definition associated with a test methodology to verify that definition.

Before giving that formal definition, we need to define what a reconfiguration is first. A reconfiguration consists of four elements, a parameter *p*, the parameter values before and after reconfiguration *v1*, *v2*, and a provided reconfiguration method *m*.

The informal defition works with two assumptions.

First, there exists an oracle that can be used to verify system correctness, involving functionality, availability and consistency. The oracle returns *ok* when the system can work correctly and *error* otherwise. Notice that the oracle has the power to explore and verify all the reconfiguration timing points.

Second, when oracle(*p, v1, v2, m_online*) returns *ok*, the return value of oracle(*p, v1, v2, m_offline*) must be *ok* as well.

Now, the informal definition can be interpreted as:

Given a parameter *p* and two reconfiguration methods *m_offline, m_online*, if $\exists$ a value pair *v1, v2* such that oracle(*p, v1, v2, m_online*) return *error* but oracle(*p, v1, v2, m_offline*) returns *ok*, then parameter *p* is *not online reconfigurable*; otherwise, parameter *p* is *online reconfigurable*.

Notice that it is not correct to simply perform oracle(*p, v1, v2, m_online*) to verify parameter reconfigurability, because we do not have the oracle to verify if parameter values are valid or not.

2.3 More Formal Definition

In this section, we will remove the two assumptions used in the informal definition.

In pratice, there is not such magical oracle to verify system correctness. On the other hand, system correctness is tested by running benchmarks and observing logs. Thus, we change the term oracle to test. We now define that test(*p, v1, v2, m*) returns *ok* when no errors are observed by benchamarks and *error* otherwise.

Now, the informal definition can be transformed as:

Given a parameter *p* and two reconfiguration methods *m_offline, m_online*, if $\exists$ a value pair *v1, v2* such that test(*p, v1, v2, m_online*) return *error* but test(*p, v1, v2, m_offline*) returns *ok*, then parameter *p* is *not online reconfigurable*; otherwise, parameter *p* is *online reconfigurable*.

So far, this definition is similar to the informal one except that we will have false negative because practical benchmarks cannot verify system correctness thoroughly and completely.

2.4 Issues with Our Informal Definition???

Actually, there is a drawback in our informal definition given above. If some parameters do not support reconfiguration neither online nor offline, then they will not be regarded as *not online reconfigurable* because test(*p, v1, v2, m*) will always return *error*. However, intuitively, not

offline reconfigurable should imply not onlone reconfig-
urable...