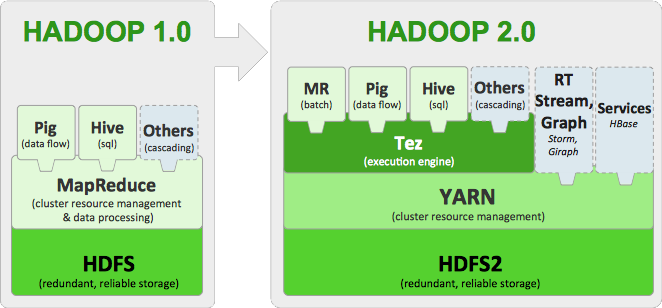
**Hadoop 2.x Architecture:**

Apache Software Foundation has released Apache Hadoop aka NextGen Hadoop 2.x that has completes makeover of architecture. Here is a glance of improvement to HDFS and MapReduce in Hadoop 2.x. There are following primary components at the core of Apache Hadoop 2 which illustrates below.

**Hadoop 1.x vs Hadoop 2.x Architecture:**

[](http://4.bp.blogspot.com/-S3nAGqw8omg/U8LHYbhSSKI/AAAAAAAABXM/92V_lFRcoPI/s1600/hadoopstack.png)

**Apache Hadoop 2.x Key Components:**

* **NameNode(HA) aka HDFS High Availability:**In Hadoop 1.0 NameNode was the single point of failure in a Cluster, resulting in data loss in case of a NameNode failure. Hadoop 2.0 Architecture supports multiple NameNodes to remove this bottleneck by using Standby NameNode.
* **HDFS Federation:** Federation enables support for multiple namespaces in the cluster to improve scalability and isolation. Federation also opens up the architecture for future, expanding the applicability of HDFS cluster to new implementations and use cases.
* **Snapshots:** This provides the ability to save the state of HDFS file system and point-in-time recovery for backup, disaster recovery and protection against use errors.
* **YARN(Yet Another Resource Negotiator) aka NextGen (MRv2):** This is next generation processing framework.

### HDFS:

**Key HDFS Features:**

* **Scale-Out Architecture** - Add servers to increase capacity
* **High Availability** - Serve mission-critical workflows and applications
* **Fault Tolerance** - Automatically and seamlessly recover from failures
* **Flexible Access** – Multiple and open frameworks for serialization and file system mounts
* **Load Balancing** - Place data intelligently for maximum efficiency and utilization
* **Tunable Replication** - Multiple copies of each file provide data protection and computational performance
* **Security** - POSIX-based file permissions for users and groups with optional LDAP integration

**NameNode High Availability:**

**Problem:** Prior to Hadoop 2.0.0, the NameNode was a single point of failure (SPOF) in an HDFS cluster. Each cluster had a single NameNode, and if that machine or process became unavailable, the cluster as a whole would be unavailable until the NameNode was either restarted or brought up on a separate machine.

**Solution:** NameNode HA feature addresses the above problems by providing the option of running two NameNodes in the same cluster, in an Active/Passive configuration. These are referred to as the Active NameNode and the Standby NameNode. Unlike the Secondary NameNode, the Standby NameNode is hot standby, allowing a fast failover to a new NameNode in the case that a machine crashes, or a graceful administrator-initiated failover for the purpose of planned maintenance. You cannot have more than two NameNodes.

**Architecture:**

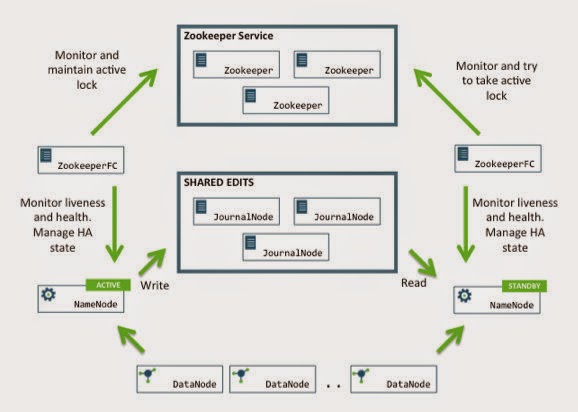
In a typical HA cluster, two separate machines are configured as NameNodes. At any point in time, exactly one of the NameNodes is in an Active state, and the other is in a Standby state. The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.

In order for the Standby node to keep its state synchronized with the Active node, the current implementation requires that the two nodes both have access to a directory on a shared storage device (eg an **NFS** mount from a NAS). This restriction will likely be relaxed in future versions.

When any namespace modification is performed by the Active node, it durably logs a record of the modification to an edit log file stored in the shared directory. The Standby node is constantly watching this directory for edits, and as it sees the edits, it applies them to its own namespace. In the event of a failover, the Standby will ensure that it has read all of the edits from the shared storage before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.

In order to provide a fast failover, it is also necessary that the Standby node have up-to-date information regarding the location of blocks in the cluster. In order to achieve this, the DataNodes are configured with the location of both NameNodes, and send block location information and heartbeats to both.

NameNode HA is achieved using existing components like ZooKeeper along with new components like a quorum of **JournalNodes** and the ZooKeeper Failover Controller (**ZKFC**) processes:

[](http://1.bp.blogspot.com/-VqMkPU3eMXY/U8LIZUhBdvI/AAAAAAAABXU/o32v9tradlk/s1600/R2.jpg)

**HDFS Federation:**

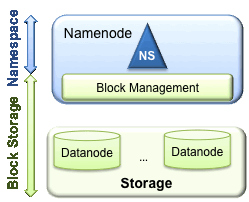
HDFS Federation improves the existing HDFS architecture through a clear separation of namespace and storage, enabling generic block storage layer. It enables support for multiple namespaces in the cluster to improve scalability and isolation. Federation also opens up the architecture, expanding the applicability of HDFS cluster to new implementations and use cases.

**Problem**: The prior HDFS architecture allows only a single namespace for the entire cluster. A single Namenode manages this namespace. HDFS Federation addresses limitation of the prior architecture by adding support multiple Namenodes/namespaces to HDFS file system.

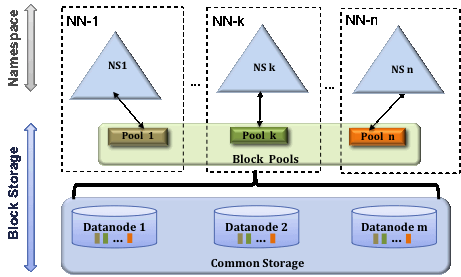
**Solution**:

**HDFS** has main two layers:

* **Namespace**
  + Consists of directories, files and blocks
  + It supports all the namespace related file system operations such as create, delete, modify and list files and directories.
* **Block Storage Service** has two parts
  + **Block Management** (which is done in Namenode)
    - Maintains membership of datanodes in the cluster by handling registrations, and periodic heart beats.
    - Processes block reports and maintains location of blocks and supports block related operations such as create, delete, modify and get block location.
    - Manages replica placement and replication of a block.
  + **Physical Storage** - Datanodes by storing blocks on the local file system and allows read/write access.

[](http://3.bp.blogspot.com/-EWNV-_ko00g/U8LIsqGWWkI/AAAAAAAABXc/roXsSBzPmkg/s1600/federation-background.gif)

In order to scale name service horizontally, federation uses multiple independent Namenodes/namespaces. The Namenodes are federated, that is, the Namenodes are independent and don’t require coordination with each other. The datanodes are used as common storage for blocks by all the Namenodes. Each datanode registers with all the Namenodes in the cluster. Datanodes send periodic heartbeats and block reports and handles commands from the Namenodes.

[](http://3.bp.blogspot.com/-Fez-VPcoHc0/U8LI6V8KVuI/AAAAAAAABXk/8Vztx5OkWWA/s1600/federation.gif)

**Block Pool:**

A Block Pool is a set of blocks that belong to a single namespace. Datanodes store blocks for all the block pools in the cluster. It is managed independently of other block pools. This allows a namespace to generate Block IDs for new blocks without the need for coordination with the other namespaces. The failure of a Namenode does not prevent the datanode from serving other Namenodes in the cluster.

A Namespace and its block pool together are called **Namespace Volume.** It is a self-contained unit of management. When a Namenode/namespace is deleted, the corresponding block pool at the datanodes is deleted. Each namespace volume is upgraded as a unit, during cluster upgrade.

**ClusterID:**

A new identifier **ClusterID** is added to identify all the nodes in the cluster. When a Namenode is formatted, this identifier is provided or auto generated. This ID should be used for formatting the other Namenodes into the cluster.

**Key Benefits:**

* **Namespace Scalability** - HDFS cluster storage scales horizontally but the namespace does not. Large deployments or deployments using lot of small files benefit from scaling the namespace by adding more Namenodes to the cluster
* **Performance** - File system operation throughput is limited by a single Namenode in the prior architecture. Adding more Namenodes to the cluster scales the file system read/write operations throughput.
* **Isolation** - A single Namenode offers no isolation in multi user environment. An experimental application can overload the Namenode and slow down production critical applications. With multiple Namenodes, different categories of applications and users can be isolated to different namespaces.

**Snapshots:**

Hadoop 2.0 HDFS allows taking snapshots of the file system. We can save the state of file system and restore it later. Snapshots can be useful for data backup, recovering from user errors and recovering from disasters.

HDFS snapshotting is pretty efficient. The actual physical data (e.g. blocks in DataNodes-anodes) is NOT copied. Snapshotting records the status of meta data. Since dealing with meta data is pretty fast, snapshotting operation is very fast. In computer science term the cost is constant O(1).

Snapshotting doesn't slow down regular HDFS operations.

HDFS uses a special directory named .snapshot for snapshot purposes. Hence the file name '.snapshot' is a reserved file name. Users can not create file or directory with the name '.snapshot'

**YARN(Yet Another Resource Negotiator or Nextgen MRv2):**

MapReduce has undergone a complete makeover in hadoop-0.23 and we now have, what we call, MapReduce 2.0 (MRv2) or YARN.

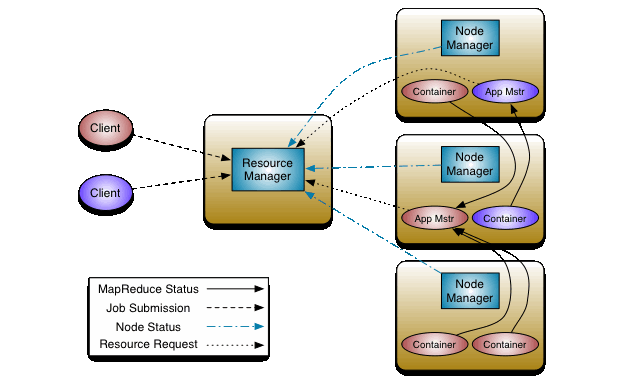
The fundamental idea of MRv2 is to split up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons.

The idea is to have a global **ResourceManager** (RM) and per-application **ApplicationMaster** (AM). An application can be a single job in the classical Map-Reduce jobs.

The **ResourceManager** and per-node slave, the **NodeManager** (NM), form the data-computation framework and form a new generic system for managing application in distributed manner. The **ResourceManager** is the ultimate authority that arbitrates resources among all the applications in the system.

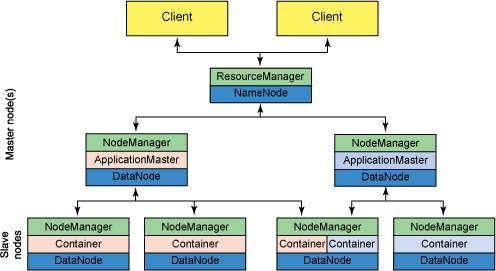
The per-application **ApplicationMaster** is, in effect, a framework specific library and is tasked with negotiating resources from the **ResourceManager** and working with the **NodeManager**(s) to execute and monitor the tasks.

Architecture of YARN and it's core components:

[](http://1.bp.blogspot.com/-dlCTOOlJX9s/U8LJmWqYG9I/AAAAAAAABXs/Nt9JYdrF4f4/s1600/yarn_architecture.gif)

The ResourceManager has two main components:

* **Scheduler**
* **ApplicationsManager**

[](http://3.bp.blogspot.com/-PzcFAXPdl1Q/U8LJt2rJjDI/AAAAAAAABX0/fXD54m758zs/s1600/yarn-arc.png)

The **Scheduler** is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based the resource requirements of the applications; it does so based on the abstract notion of a resource **Container** which incorporates elements such as memory, cpu, disk, network etc. In the first version, only memory is supported.

The **Scheduler** has a pluggable policy plug-in, which is responsible for partitioning the cluster resources among the various queues, applications etc. The current Map-Reduce schedulers such as the **CapacityScheduler** and the **FairScheduler** would be some examples of the plug-in.

The **CapacityScheduler** supports hierarchical queues to allow for more predictable sharing of cluster resources. **CapacityScheduler** is by default scheduler for Hadoop 2.x.

The **ApplicationsManager** is responsible for accepting job-submissions, negotiating the first **container** for executing the application specific **ApplicationMaster** and provides the service for restarting the **ApplicationMaster** container on failure if any.

The **NodeManager** is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the **ResourceManager/Scheduler**.

The per-application **ApplicationMaster** has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

MRV2 maintains API compatibility with previous stable release (Hadoop-1.x). This means that all Map-Reduce jobs should still run unchanged on top of MRv2 with just a recompile.

|  |  |  |
| --- | --- | --- |
| **Sl No** | **Hadoop1** | **Hadoop2** |
| **1** | Supports MapReduce (MR) processing model only. Does not support non-MR tools | Allows to work in MR as well as other distributed computing models like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors. |
| **2** | MR does both processing and cluster-resource management. | YARN (Yet Another Resource Negotiator) does cluster resource management and processing is done using different processing models. |
| **3** | Has limited scaling of nodes. Limited to 4000 nodes per cluster | Has better scalability. Scalable up to 10000 nodes per cluster |
| **4** | Works on concepts of slots – slots can run either a Map task or a Reduce task only. | Works on concepts of containers. Using containers can run generic tasks. |
| **5** | A single Namenode to manage the entire namespace. | Multiple Namenode servers manage multiple namespaces. |
| **6** | Has Single-Point-of-Failure (SPOF) – because of single Namenode- and in the case of Namenode failure, needs manual intervention to overcome. | Has to feature to overcome SPOF with a standby Namenode and in the case of Namenode failure, it is configured for automatic recovery. |
| **7** | MR API is compatible with Hadoop1x. A program written in Hadoop1 executes in Hadoop1x without any additional files. | MR API requires additional files for a program written in Hadoop1x to execute in Hadoop2x. |
| **8** | Has a limitation to serve as a platform for event processing, streaming and real-time operations. | Can serve as a platform for a wide variety of data analytics-possible to run event processing, streaming and real-time operations. |
| **9** | A Namenode failure affects the stack. | The Hadoop stack – Hive, Pig, HBase etc. are all equipped to handle Namenode failure. |
| **10** | Does not support Microsoft Windows | Added support for Microsoft windows |

**DIFFERENCE BETWEEN HADOOP 1.X AND HADOOP 2.X**

**Detail Description:**

Now, let us see the above details on how Hadoop1 and Hadoop2 are different in brief.

### Scalability

In Hadoop2.x with the help of YARN  architecture, we can run larger clusters than Hadoop v1. Hadoop v1 hits scalability bottlenecks in the region of 4,000 nodes and 40,000 tasks, deriving from the fact that the job tracker has to manage both jobs and tasks. YARN overcomes these limitations by virtue of its split resource manager/application master architecture: It is designed to scale up to 10,000 nodes and 100,000 tasks.

In contrast to the jobtracker, each instance of an application  – here, a MapReduce job – has a dedicated application master, which runs for the duration of the application. This model is actually closer to the original GFS paper, which describes how a master process is started to coordinate map and reduce tasks running on a set of workers.

### Ability to run non-MapReduce – jobs

In Hadoop1.x, we can only run MapReduce framework jobs to process the data which is stored in HDFS. We couldn’t had the opportunity to run other applications than MapReduce in the HDFS cluster. Thus, Hadoop2.x came up with new framework YARN which provides the ability to run non-MapReduce jobs like Spark, Hama, Giraph, Message Passing Interface) MPI & HBase coprocessors.

### Namenode High Availability

Previously, in Hadoop1.x we had single namenode which maintained a directory tree of HDFS files and tracked where data was stored in the cluster.  If the Namenode is down due to some unplanned event such as a machine crash, the whole Hadoop cluster will be down as well.

Hadoop2.x comes with the solution for this problem, which allows users to configure clusters with redundant namenodes, removing the chance that a lone namenode will become a single point of failure within a cluster.

### Native Windows Support

Hadoop was originally developed to support the UNIX family of operating systems. With Hadoop2, the Windows operating system is natively supported. This extends the reach of Hadoop significantly to a sizable Windows Server market.

### Beyond Batch Oriented application

Hadoop goes beyond Batch oriented nature in its version 2.0 and now can run interactive, streaming application also.

#### **Utilization**

In MapReduce v1, each tasktracker is configured with a static allocation of fixed-size “slots”, which are divided into map slots and reduce slots at configuration time. A map slot can only be used to run a map task, and a reduce slot can only be used for a reduce task. In YARN, a nod manager manages a pool of resources, rather than a fixed number of designated slots.

MapReduce running on YARN will not hit the situation where a reduce task has to wait because only map slots are available in the cluster, which can happen in MapReduce v1. If the resources to run the task are available, then the application will be eligible for them. Furthermore, resources in YARN are fine grained, so an application can make a request for what it needs, rather than for an indivisible slot, which may be too big (which is wasteful of resources) or too small (which may cause a failure) for the particular task. Multitenancy in some ways, the biggest benefit of YARN is that it opens up Hadoop to other types of distributed application beyond MapReduce

MapReduce is just one YARN application among many. It is even possible for users to run different versions of MapReduce on the same YARN cluster, which makes the process of upgrading MapReduce more manageable.

So this is the main differences between Hadoop1 and Hadoop architecture.