Hadoop 1.x vs 2.x vs 3.x

Hadoop Version Comparison

Feature	Hadoop 1.x	Hadoop 2.x	Hadoop 3.x
Resource Management	**MapReduce (JobTracker)** Tightly coupled resource and job scheduling	Introduced **YARN** (Yet Another Resource Negotiator) Decouples resource management, supports multi-frameworks (e.g., Spark, Flink)	Enhanced YARN Supports heterogeneous resources (GPU/FPGA)
HDFS High Availability	**Single NameNode, Master/Slave topology → SPOF; No HA support**	**Active/Standby NameNodes** with QJM Metadata synchronization	**Multiple NameNodes** with Raft protocol Improved fault tolerance
Storage Efficiency	3x replication(3副本机制)	3x replication	**Erasure Coding(纠删码)** Reduces storage overhead to 50%
Single Point of Failure	JobTracker + NameNode (SPOF)	NameNode HA eliminates SPOF	Further reduced with multiple NameNodes
Scalability	≤ 4,000 nodes	≤ 10,000 nodes	Supports ultra-large clusters
Compute Framework Support	**Only MapReduce**	**Multiple frameworks** (**Spark**, **Flink**, etc.)	Enhanced compatibility (e.g., Kubernetes)
Java Version	Java 6/7	Java 7/8	**Java 8+**
Ecosystem	Limited (MapReduce-centric)	Broad (supports **real-time streaming**, etc.)	Optimized (**Docker**/**GPU** support)
Key Innovations	Basic **MapReduce** and **HDFS**	**YARN**, **HDFS Federation**	**Erasure Coding**, **multi-NameNode**, resource type flexibility

- **Hadoop 1.x**：以HDFS和MapReduce作为核心组件，奠定了其在大数据领域的基础。
- **Hadoop 2.x**：引入YARN（Yet Another Resource Negotiator），对架构进行重大改进，使得资源管理更加高效。

- **Hadoop 3.x**：在前两个版本的基础上进行了多项改进，包括增加存储效率、提升集群规模、支持云平台和容器化。

Details

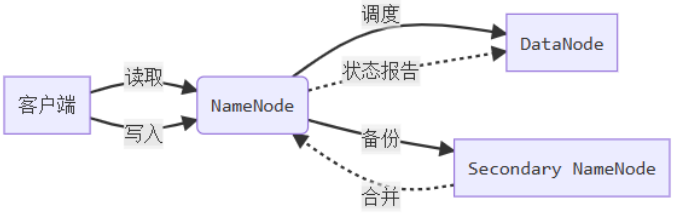
Hadoio 1.x

HDFS

Hadoop Distributed File System（HDFS）是Hadoop项目的核心组件之一，它被设计用来存储大量的数据集。HDFS的设计灵感来自于Google的GFS（Google File System），目的是在普通硬件上实现高吞吐量的数据访问，特别适合于运行大型应用。HDFS具有高容错性的特点，能够自动处理失败节点。

HDFS的主要组件包括：

- NameNode：它是一个管理文件系统命名空间的节点，负责管理文件系统的元数据。它记录了每个文件中各个块所在的DataNode节点，并且知道如何将文件分割成块。
- DataNode：这是存储实际数据的节点，负责处理文件系统客户端的读写请求，并且在NameNode的调度下创建、删除和复制块。
- Secondary NameNode：它并不是一个备份NameNode，而是帮助合并编辑日志和文件系统镜像，并且定期减轻NameNode的工作负担。



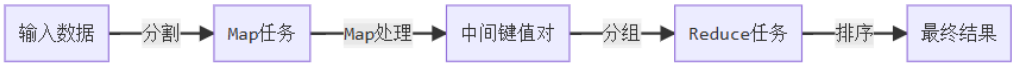
MapReduce

MapReduce是一个编程模型，用于处理和生成大数据集。MapReduce的目的是简化编写可以并行处理大规模数据集的程序。它由两部分组成：Map阶段和Reduce阶段。

- Map阶段：这一阶段主要处理输入数据，将其分割成独立的块，并进行处理，然后输出中间结果。
- Reduce阶段：在Map阶段完成后，将中间结果汇总，进行排序和合并，并输出最终结果。

MapReduce编程模型的执行流程如下：

1. 输入数据被分割成独立的数据块，每个数据块由一个Map任务处理。
2. Map任务读取数据块，并执行用户定义的Map函数，该函数处理数据并输出一系列的键值对。
3. 生成的键值对被分组，键相同的值被汇总在一起。
4. Reduce任务对分组后的键值对进行排序，然后每个键对应的所有值被传递给Reduce函数。
5. Reduce函数处理这些值，并生成最终的输出结果。



Master/Slave topology

2.2.1 Master/Slave架构及其局限性

Hadoop 1.x采用的是Master/Slave架构，即一个主节点（NameNode）管理多个从节点（DataNode）。这种设计简单直观，易于理解和实现。然而，随着系统规模的增长，这种架构会遇到一些局限性：

1. NameNode的瓶颈：所有的元数据操作都必须通过NameNode，随着数据量的增加，NameNode的内存和CPU成为性能瓶颈。
2. 单点故障：NameNode是整个Hadoop集群的单点故障点，一旦NameNode出现故障，整个集群将无法使用。
3. 容量限制：单个NameNode能够管理的文件数量和数据量有限，不适合扩展到很大的规模。

Hadoop 2.x

YARN

3.1 YARN的引入与核心变更

3.1.1 YARN架构和资源管理优化

从Hadoop 2.x版本开始，YARN（Yet Another Resource Negotiator）的引入彻底改变了Hadoop的资源管理和任务调度方式。在Hadoop 1.x中，MapReduce同时扮演着作业调度和资源管理的角色，这限制了集群的扩展性和资源利用率。YARN的出现，将资源管理和作业调度分离成两个不同的服务：ResourceManager（RM）和ApplicationMaster（AM）。

ResourceManager负责管理集群中的计算资源，并根据应用需求分配资源。它由两个主要组件构成：调度器（Scheduler）和应用程序管理器（ApplicationManager）。调度器负责分配集群中的资源块给各个应用程序，而应用程序管理器则负责接收用户提交的应用程序，并在系统中为它们启动相应的ApplicationMaster。

代码示例：启动YARNResourceManager

```
start-yarn.sh
```

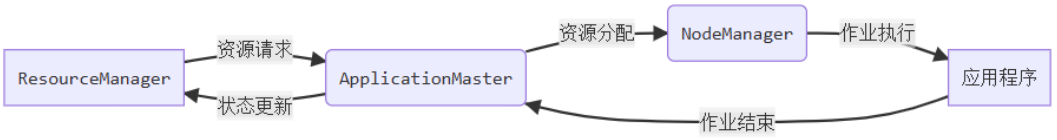
在启动YARNResourceManager后，每个提交的应用程序都会有一个对应的ApplicationMaster实例，负责与ResourceManager协商资源，并监控应用程序在节点管理器(NodeManager)上的执行情况。NodeManager是运行在每个数据节点上的组件，负责监控节点的资源使用情况（如CPU、内存、磁盘、网络等）并报告给ResourceManager。

3.1.2 Hadoop 2.x的生态系统变化

YARN的引入不仅改变了Hadoop集群的架构，也促进了Hadoop生态系统的发展。基于YARN，开发者可以运行各种不同类型的计算框架和应用程序，而不仅限于MapReduce。这样的变化使得Hadoop集群能够承载更广泛的数据处理工作负载。

生态系统中的其他项目，如Hive、Pig、Spark等，都开始与YARN集成，允许它们作为独立的应用程序运行在YARN之上。这一变化为Hadoop带来了更大的灵活性和扩展性，它不再只是一个批处理的存储和分析系统，而逐渐演变成一个支持多种计算模型的通用数据处理平台。

mermaid格式流程图：YARN资源管理流程



以上流程图展示了在YARN架构下，应用程序是如何申请资源，并由ResourceManager和NodeManager共同协作完成任务的过程。

Hadoop 3.x

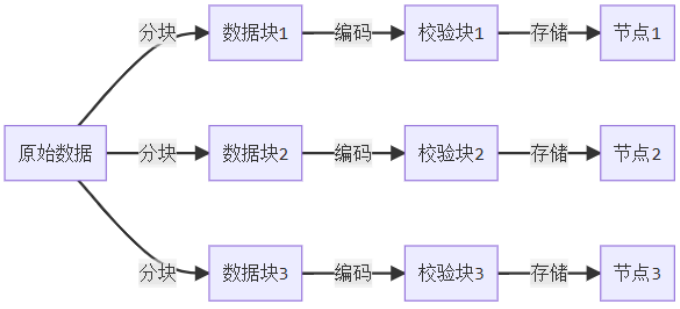
Hadoop Erasure Coding

4.1.1 新增的存储能力：HDFS Erasure Coding

Hadoop 3.x 对 HDFS（Hadoop Distributed File System）存储层进行了改进，引入了Erasure Coding（EC）技术，相较于传统的复制策略，EC 提供了更好的存储效率，尤其是在存储成本和空间利用率上。

在传统的复制策略中，系统会将数据副本存放在多个节点上，以确保数据的可靠性和容错能力。然而，这种策略导致存储利用率低下，尤其对于大规模集群而言，成本和空间消耗是主要的考虑因素。

通过Erasure Coding技术，可以将数据分割成块，并对这些数据块进行编码，生成一组校验块，之后将数据块和校验块分别存储在不同的节点上。当部分数据丢失时，系统可以通过剩余的数据块和校验块重新构造丢失的数据。这一策略相较于三副本策略，可以显著减少所需的存储空间，从而提高存储效率。



参数说明：

- **数据块**：原始数据分割后形成的块。
- **校验块**：编码后产生的额外数据块，用于数据恢复。
- **节点**：数据块和校验块存放的位置。