A basic Apache Hadoop YARN system has two core components:

▪ The Hadoop Distributed File System for storing data, which will be referred to as HDFS.

▪ Hadoop YARN for implementing applications to process data.

Apache Hadoop provides a basis for large scale MapReduce processing and has spawned a big data ecosystem of tools, applications, and vendors. While MapReduce methods enable the users to focus on the problem at hand rather than the underlying processing mechanism, they do limit some of the problem domains that can run in the Hadoop framework.

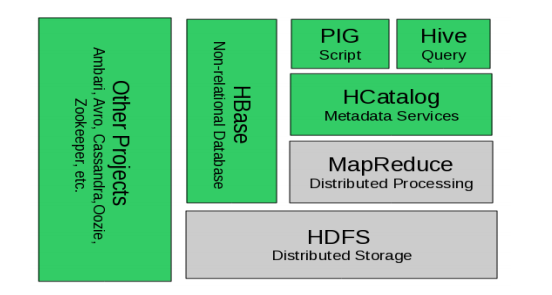
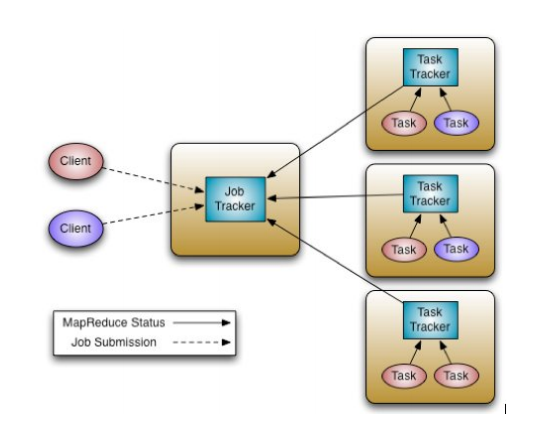
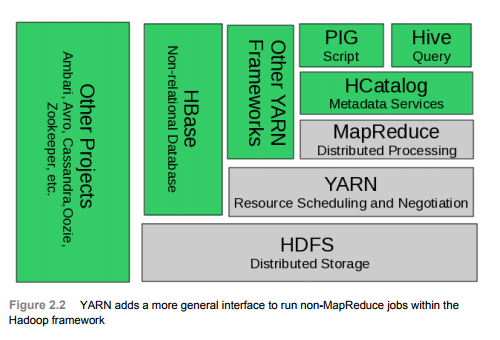


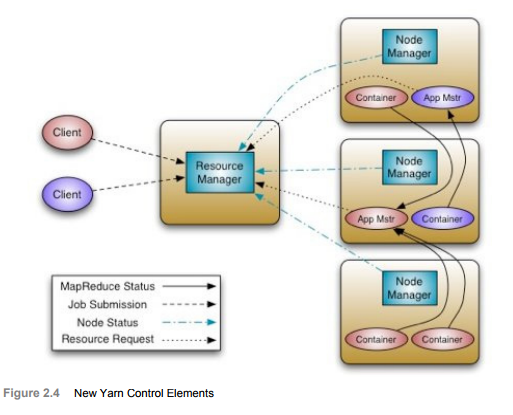
Figure 2.1 The Hadoop 1.0 ecosystem, MapReduce and HDFS are the core components, while other are built around the core.



In Hadoop 1 ,the JobTracker views the cluster as composed of nodes (managed by individual TaskTrackers) with distinct map slots and reduce slots, which are not fungible. Utilization issues occur because maps slots might be ‘full’ while reduce slots are empty (and vice-versa). Improving this situation is necessary to ensure the entire system could be used to its maximum capacity for high utilization and applying resources when needed.

**Hadoop 2.0**





**Apache Hadoop YARN** The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker, in other words resource management and job scheduling/monitoring, into separate daemons: a global ResourceManager and per-application ApplicationMaster (AM). The ResourceManager and per-node slave, the NodeManager (NM), form the new, and generic, operating system for managing applications in a distributed manner.

YARN brings with it several new services that separate it from the standard Hadoop MapReduce model. A new **ResourceManger** acting as a pure resource scheduler is the sole arbitrator of cluster resources. User applications, including MapReduce jobs, ask for specific resource requests via the new **ApplicationMaster** component, which in in-turn negotiates with the **ResourceManager** to create an **application container** within the cluster. By incorporating MapReduce as a YARN framework, YARN also provides full backward compatibility with existing MapReduce tasks and applications.

Apache Hadoop YARN The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker, in other words resource management and job scheduling/monitoring, into separate daemons: a global ResourceManager and per-application ApplicationMaster (AM). The ResourceManager and per-node slave, the NodeManager (NM), form the new, and generic, operating system for managing applications in a 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 entity and is tasked with negotiating resources from the Resource

Components in Yarn:

**Resource Manager** the YARN ResourceManager is primarily a pure scheduler. It is strictly limited to arbitrating available resources in the system among the competing applications. It optimizes for cluster utilization (keeps all resources in use all the time) against various constraints such as capacity guarantees, fairness, and SLAs. To allow for different policy constraints the ResourceManager has a pluggable scheduler that enables different algorithms such as capacity and fair scheduling to be used as necessary

**ApplicationMaster** An important new concept in YARN is the ApplicationMaster. The ApplicationMaster is, in effect, an instance of a framework-specific library and is responsible for negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the containers and their resource consumption. It has the responsibility of negotiating appropriate resource containers from the ResourceManager, tracking their status and monitoring progress. The ApplicationMaster design enables YARN to offer the following important new features: ▪ Scale: The Application Master provides much of the functionality of the traditional ResourceManager so that the entire system can scale more dramatically. Simulations have shown jobs scaling to 10,000 node clusters composed of modern hardware without significant issue. As a pure scheduler the ResourceManager does not, for example, have to provide fault-tolerance for resources across the cluster. By shifting fault tolerance to the ApplicationMaster instance, control becomes local and not global. Furthermore, since there is an instance of an ApplicationMaster per application, the ApplicationMaster itself isn’t a common bottleneck in the cluster. ▪ Open: Moving all application framework specific code into the ApplicationMaster generalizes the system so that we can now support multiple frameworks such as MapReduce, MPI and Graph Processing. These features were the result of some key YARN design decisions: ▪ Move all complexity (to the extent possible) to the ApplicationMaster while providing sufficient functionality to allow application-framework authors sufficient flexibility and power. ▪ Since it is essentially user-code, do not trust the ApplicationMaster(s). In other words. no ApplicationMaster is a privileged service. ▪ The YARN system (ResourceManager and NodeManager) has to protect itself from faulty or malicious ApplicationMaster(s) and resources granted to them at all costs.