**Architecture of Apache Hadoop Yarn**

The fundamental idea of YARN is to split up the major responsibilities of the JobTracker, i.e. 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.

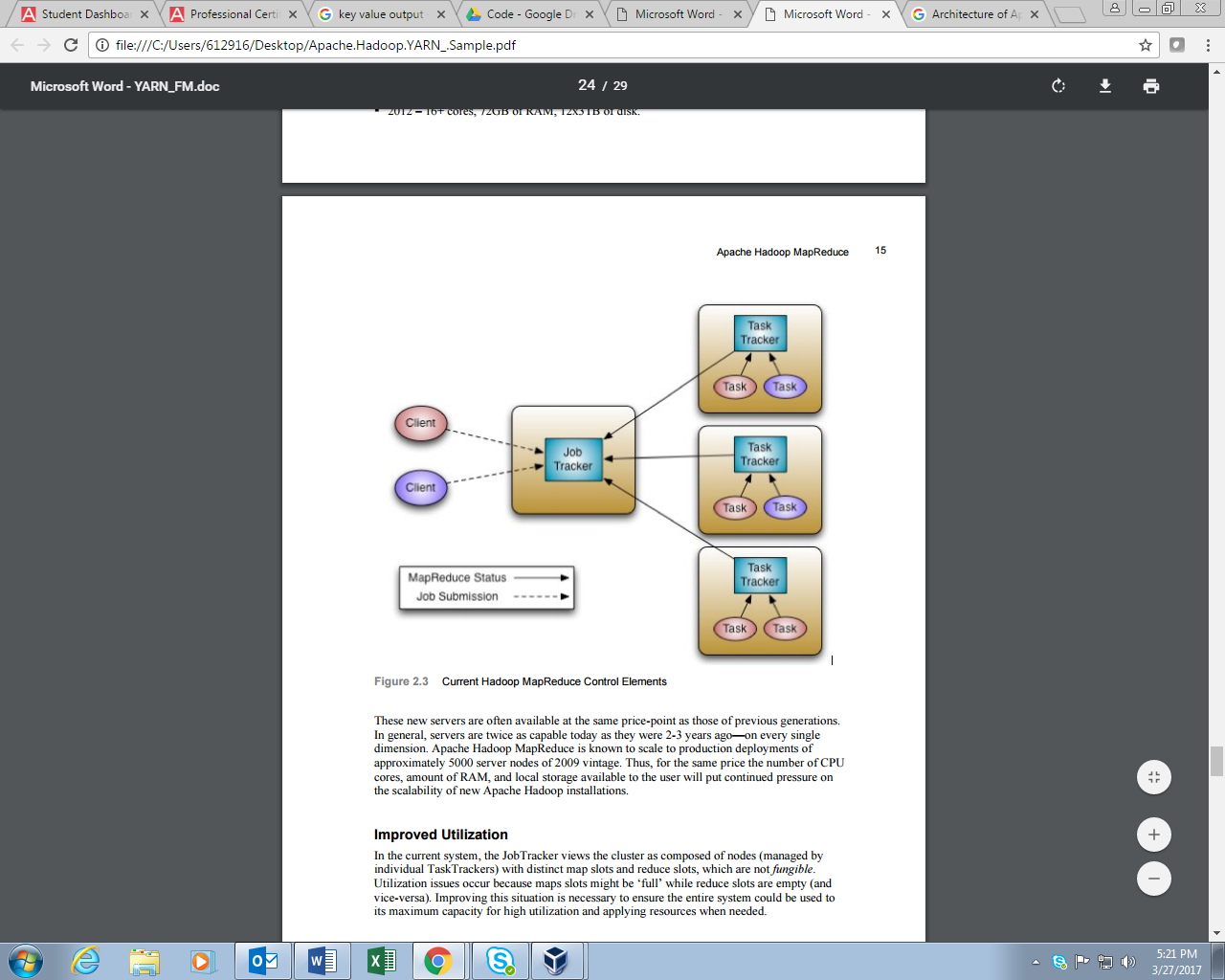
The resource manager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is, in effect, a framework specific entity and is tasked with negotiating resources from ResourceManager and working with the NodeManager to execute and monitor the component tasks.

The ResourceManager has a pluggable scheduler component, which is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The scheduler is a pure scheduler in the sense that it performs no monitoring or tracking of status for the application, offering no guarantees on restarting failed tasks either due to application failure or hardware failures. The scheduler performs its scheduling function based on the resource requirements of an application by using the abstract notion of a resource container, which incorporates resource dimensions such as memory, CPU, disk, network etc.

The NodeManager is the per-machine slave, which is responsible for launching the applications’ containers, monitoring their resource usage (CPU, memory, disk, network), and reporting the same to the ResourceManager.

The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the scheduler, tracking their status and monitoring for progress. The application Manager itself runs as a normal container from the system’s perpective.

One of the crucial implementation details for MapReduce within the new YARN system is the reuse of the existing MapReduce firework without any major changes. This step was very important to ensure compatibility for existing MapReduce applications and users.



**Architecture of Apache YARN**

**YARN Components:**

YARN brings new components into the APACHE Hadoop workflow. These components provide a finer grain of control for the end user and at the same time offer more advanced capabilities to the Hadoop ecosystem.

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 against various constraints such as capacity guarantees, fairness, and SLA’s. To allow for different policy constraints the Resource Manager has a pluggable scheduler that enables different algorithms such as capacity and fair scheduling to be used as necessary.

APPLICATION MASTER (AM):

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:

1. 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. Since there is an instance of an ApplicationMaster per application, the ApplicationMaster itself isn’t a common bottleneck in the cluster.
2. 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:

1. Move all complexity (to the extent possible) to the ApplicationMaster while providing sufficient functionality to allow application-framework authors sufficient flexibility and power.
2. Since it is essentially user-code, do not trust the ApplicationMaster(s). In other words, no ApplicationMaster is a privileged service.
3. The YARN system (ResourceManager and NodeManager) has to protect itself from faulty or malicious ApplicationMaster(s) and resources granted to them at all costs.

Resource Model

YARN supports a very general resource model for applications. An application (via the ApplicationMaster) can request resources with highly specific requirements such as:

1. Resource-name (including hostname, rack name and possibly complex network topologies)
2. Amount of Memory
3. CPUs (number/type of cores)
4. Eventually resources like disk/network I/O, GPUs, etc.

Resource Request and Containers:

YARN is designed to allow individual applications to utilize cluster resources in a shared, secure and multi-tenant manner. It also remains aware of cluster anatomy in order to efficiently schedule and optimize data access.

In order to meet those goals, the central Scheduler in the Resource Manager has extensive information about an application’s resource needs, which allows it to make better scheduling decisions across all application’s resource needs, which allows it to make better scheduling decisions across all applications in the cluster. This leads to the Resource request by granting a container, which satisfies the requirements laid out by the Application Master in the initial Resource Request.

These components are described as follows:

1. Resource-name is either hostname, rack name or to indicate no preference, Future plans may support even more complex topologies for virtual machines on a host, more complex networks etc.
2. Priority is intra-application priority for this request.
3. Resource-requirement is required capabilities such as memory, CPU etc.
4. Number of containers is just a multiple of such containers.

Essentially, the Container is the resource allocation, which is successful result of the Resource Manager granting a specific Resource Request. A container grants rights to an application to use a specific amount of resources on a specific host.

The application Master has to take the Container and present it to the Node Manager managing the host, on which the container was allocated, to use the resources for launching its tasks. For security reasons, the Container allocation is verified, to ensure that Application Masters cannot fake allocations in the cluster.

CONTAINER SPECIFICATION:

While a container, as described above, is merely a right to use a specified amount of resources on a specific machine in the cluster, the Application Master has to provide considerable more information to the Node Manager to actually launch the container. YARN allows applications to launch any process and unlike existing Hadoop MapReduce, it isn’t limited to JAVA applications.

The YARN Container launch specification API is platform agnostic and contains:

1. Command line to launch the process the process within the container.
2. Environment variables.
3. Local resources necessary on the machine prior to launch, such as jar, shared-objects, auxiliary data files etc.
4. Security related tokens.

This design allows the Application Master to work with the Node Manager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged VM (Virtual Machines).

WRAP UP:

The release of Apache Hadoop YARN provides many new capabilities to the existing Hadoop big data ecosystem. While the scalable MapReduce paradigm has enabled previously intractable problems to the efficiently managed on large clustered systems, YARN provides a framework for managing both MapReduce and non-MapReduce tasks of greater size and complexity. YARN provides the framework to apply low cost commodity hardware to virtually any big data problem.