Both systems have the same goal: allowing you to share a large cluster of machines between different frameworks.   
  
For those who don't know, NextGen MapReduce is a project to factor the existing MapReduce into a generic layer that handles distributed process execution and resource scheduling (this system is called YARN) and then implement MapReduce as an "application" on top of this.  
  
Mesos was originally an academic research project with a very similar goal. They created a system which could run a patched version of Hadoop, MPI and other things. This has grown into an Apache Incubator project in its own right.  
  
I have been looking into these two a bit because we would love something like this at LinkedIn, and the nature of these things is that you really only want one (since you want to run everything on it). So at the moment we don't have any real experience running stuff on top of either of these, but here is what I have pieced together (may be wrong in places):

1. Nextgen MapReduce (aka YARN) is primarily written in java with bits of native code. Mesos is primarily written in C++.
2. YARN only handles memory scheduling (e.g. you request x containers of y MB each), but with plans to extend it to other resources. I believe Mesos handles both memory and CPU scheduling, but I don't know the details. In practice I think the OS handles CPU scheduling pretty well so I am not sure that would help our use cases. Supporting some kind of disk space and disk I/O scheduling and enforcement would be super cool, but I don't think either do that (yet).
3. Mesos uses Linux container groups ([http://lxc.sourceforge.net](http://lxc.sourceforge.net/)), and YARN uses simple unix processes. Linux container groups are a stronger isolation but may have some additional overhead.
4. The resource request model is weirdly backwards in Mesos. In YARN you (the framework) request containers with a given specification and give locality preferences. In Mesos you get resource "offers" and choose to accept or reject those based on your own scheduling policy. The Mesos model is a arguably more flexible, but seemingly more work for the person implementing the framework.
5. YARN is a pretty epic chunk of code, including all kinds of things right down to its own web framework. It is about 3x as much code as Mesos.
6. YARN integrates something similar to the pluggable schedulers everyone knows and loves/hates in Hadoop. So if you are used to the capacity scheduler, hierarchical queues, and all that, you can get something similar. I don't think the Mesos scheduling capabilities are quite as robust (they list hierarchical scheduling on their roadmap).
7. YARN integrates with Kerberos and essentially inherits the Hadoop security architecture. I don't think Mesos attempts to deal with security.
8. YARN directly handles rack and machine locality in your requests, which is convenient. In Mesos you can implement this, but it is less out of the box.
9. Mesos is much more mature as a project at this point. It is a standalone thing, with great documentation, and good starter examples. YARN exists only on hadoop trunk (and some feature branches) in the mapreduce directory, and the docs are super sparse. That said, the Hadoop guys have been really awesome at helping us get started with YARN (thanks [Arun](https://www.quora.com/profile/Arun-C-Murthy)!) and they seem really committed to making sure it works as a general purpose framework, not just for Hadoop. There seems to be a lot of momentum, it is just early.
10. YARN is going to be the basis for Hadoop MapReduce going forward, so if you have a big Hadoop cluster and want to be able to run other stuff on it, that is likely appealing and will probably work more transparently than Mesos.
11. YARN was written by the Yahoo/HortonWorks Hadoop team which has should know a thing or two about multi-tenancy and very large-scale cluster computing. YARN is not yet in a stable Hadoop release so I am not sure how much actual testing it has had or the extent of deployment internally at Yahoo. Regardless, if/when the YARN team is able to get the majority of the worlds Hadoop clusters successfully running on top of YARN, that will likely get the project to a level of hardening that will be hard to compete with.
12. Mesos ships with a number of out-of-the-box frameworks ported to it. This somewhat helps to validate the generality of their framework, but i don't know how much of a hack the various ports of things to it are.

Here are a few pointers for folks trying to find out more about Mesos:

1. Docs: [http://www.mesosproject.org/docu...](http://www.mesosproject.org/documentation.html)
2. Papers: [http://www.mesosproject.org/rese...](http://www.mesosproject.org/research.html)
3. Sample framework implementations: [https://github.com/mesos/mesos/t...](https://github.com/mesos/mesos/tree/master/src/examples)

Here are some pointers on YARN:

1. Master JIRA: [https://issues.apache.org/jira/b...](https://issues.apache.org/jira/browse/MAPREDUCE-279)
2. Article on the new resource scheduler: [http://developer.yahoo.com/blogs...](http://developer.yahoo.com/blogs/hadoop/posts/2011/03/mapreduce-nextgen-scheduler)
3. Design document for YARN. This is really essential for understanding their terminology of application masters, resource manager, etc. Before we found this, just looking at code, we were lost. [https://issues.apache.org/jira/s...](https://issues.apache.org/jira/secure/attachment/12486023/MapReduce_NextGen_Architecture.pdf)
4. Spark, an iterative machine learning framework, has been ported to YARN, and serves as a great example of how to do this: <https://github.com/mesos/spark-yarn>

There is a thread on the Mesos mailing list that discusses differences further: [http://mail-archives.apache.org/...](http://mail-archives.apache.org/mod_mbox/incubator-mesos-dev/201107.mbox/%3CBANLkTim=L9eJ8FJ7pkjHiSb8SUZLZMQuyA@mail.gmail.com%3E)

|  |  |  |
| --- | --- | --- |
| Sno. | Yarn | Mesos |
| 1 | In java | C++ |
| 2 | When job request comes into the Yarn resource manager, it evaluates all the resources available and places the job accordingly. Thus it is a **monolithic scheduler** (Monolithic schedulers are a single process entity, that make scheduling decisions and deploy jobs to be scheduled. | When a job comes into execution, the job request comes into Mesos master and Mesos determines the resources that are available and sends the request to the framework. This allows the framework to determine what is the best fit for a job that’s needed to be run. Thus, it is **non-monolithic scheduler**(it is two way process entity, that makes scheduling decision and deploy job to the scheduler). |
| 3 | In YARN, it is mainly memory scheduling, i.e. pull based scheduling. | In Mesos, it is a memory and CPU scheduling, i.e. push based scheduling. |
| 4 | It is less scalable because it is a monolithic scheduler. | Due to non-monolithic scheduler, Mesos is highly scalable. |
| 5 | It can safely manage the Hadoop job but it is not capable of managing the entire data center. | If we want to manage data center as a whole, **Apache Mesos**can manage every single resource in the data center. |
| 6 | Here we can run YARN on Mesos (Myriad). | Here we get Low-level abstraction. |
| 7 | If a YARN **resource manager fails,** it recovers from its own failure by restoring its state from a persistent store on initialization; it kills all the containers running in the cluster after the recovery process is complete. While when a **node manager fails**, the resource manager detects it by timing out its heartbeat response, marks all the containers running on that node as killed, and reports the failure to all running Application Master. If the fault is transient, the [YARN node manager](http://data-flair.training/blogs/hadoop-yarn-node-manager-tutorial-guide/)will re-synchronize with the resource manager, clean up its local state, and continue. | It provides fault tolerance at each step**. At master level,** to make master fault tolerant, Zookeeper monitors all the nodes in the master cluster and if the hot master node fails, it elects the new Master. In order to make **framework**fault tolerant, two or more schedulers are registered with the master. In case if one scheduler fails, the master will notify another scheduler. If **the slave** process fails, the task continues running and when the master restarts the slave process because it is not responding to messages, the restarted slave process will use the check pointed data to recover state and to reconnect with executors/tasks. |
| 8 | Here each time the Framework asks a container with specification and preferences, so lots of information is required to be passed. | When Framework asks a container, it gets to choose a resource. Thus, very minimal information is just needed. |
| 9 | YARN is responsible to set up the "one-time framework" for each job. | Overall, we submit jobs directly to the frameworks and Mesos is not aware of the jobs; we are responsible to set up frameworks. |
|  |  |  |

Reference quora

http://www.agildata.com/apache-spark-cluster-managers-yarn-mesos-or-standalone/

<https://www.quora.com/How-does-YARN-compare-to-Mesos-What-are-the-key-differences>

https://data-flair.training/blogs/hadoop-yarn-resource-manager/

https://www.oreilly.com/ideas/a-tale-of-two-clusters-mesos-and-yarn

https://dzone.com/articles/workload-and-resource-management-yarn-mesos-and-my

<https://data-flair.training/blogs/apache-spark-cluster-managers-tutorial/>