LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

IMPROVE PERFORMANCE AND RELIABILITY WITH MYSQL REPLICATION

AN OVERVIEW OF THE TAILS DESKTOP FOR SECURITY

APRIL 2014 | ISSUE 240 | www.linuxjournal.com

HIGH-PERFORMANCE COMPUTING

A PROCESS FOR MANAGING AND CUSTOMIZING HPC OPERATIONS

LIGHTWEIGHT VIRTUALIZATION WITH LINUX-BASED CONTAINERS

JOB SCHEDULING IN HADOOP WITH YARN

WATCH: ISSUE OVERVIEW

How YARN Changed Hadoop Job Scheduling

Got Cluster Scheduling? It's old hat for HPC admins, but have you ever wondered how Hadoop does workload management and job scheduling?

ADAM DIAZ

Scheduling means different things depending on the audience. To many in the business world, scheduling is synonymous with workflow management. Workflow management is the coordinated execution of a collection of scripts or programs for a business workflow with monitoring, logging and execution guarantees built in to a WYSIWYG editor. Tools like Platform Process Manager come to mind as an example. To others, scheduling is about process or network scheduling. In the distributed computing world, scheduling means job scheduling, or more correctly, workload management.

Workload management is not only about how a specific unit of work is submitted, packaged and scheduled, but it's also about how it runs, handles failures and returns results. The HPC definition is fairly close to the Hadoop definition of scheduling. One interesting way that HPC scheduling and resource management cross paths is within the Hadoop on Demand project. The Torque resource manager and Maui Meta Scheduler both were used for scheduling in the Hadoop on Demand project during Hadoop's early days at Yahoo.

This article compares and contrasts the historically robust field of HPC workload management with the rapidly evolving field of job scheduling happening in Hadoop today.

Both HPC and Hadoop can be called distributed computing, but they diverge rapidly architecturally. HPC is a typical share-everything architecture with compute nodes sharing common storage. In this case, the data for each job has to be moved to the node via the shared storage system. A shared storage layer makes writing job scripts a little easier, but it also injects the need for more expensive storage technologies. The share-everything paradigm also creates an ever-increasing demand on the network with scale. HPC centers quickly realize they must move to higher speed networking technology to support parallel workloads at scale.

Hadoop, on the other hand, functions in a share-nothing architecture, meaning that data is stored on individual nodes using local disk. Hadoop moves work to the data and leverages inexpensive and rapid local storage (JBOD) as much as possible. A local storage architecture scales nearly linearly due to the proportional increase in CPU, disk and I/O capacity as node count increases. A fiber network is a nice option with Hadoop, but two bonded 1GbE interfaces or a single 10GbE in many cases is fast

enough. Using the slowest practical networking technology provides a net savings to a project budget.

From a Hadoop philosophy, funds really should be allocated for additional data nodes. The same can be said about CPU, memory and the drives themselves. Adding nodes is what makes the entire cluster both more parallel in operation as well as more resistant to failure. The use of mid-range componentry, also called commodity hardware is what makes it affordable.

Until recently, Hadoop itself was a paradigm restricted mainly to MapReduce. Users have attempted to stretch the model of MapReduce to fit an ever-expanding list of use cases well beyond its intended roots. The authors of Hadoop addressed the need to grow Hadoop beyond MapReduce architecturally by decoupling the resource management features built in to MapReduce from the programming model of MapReduce.

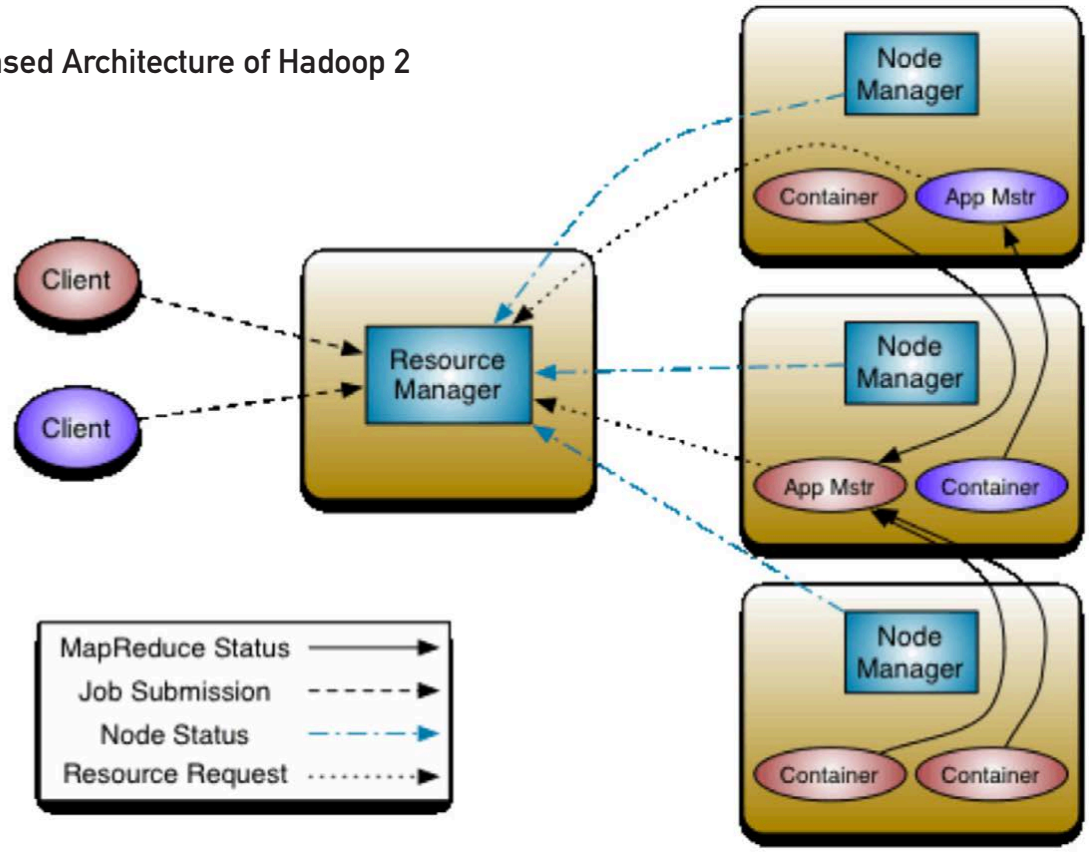
The new resource manager is referred to as YARN. YARN stands for Yet Another Resource Negotiator and was introduced in the ASF JIRA MAPREDUCE-279. The YARN-based architecture of Hadoop 2 allows for alternate programming paradigms within Hadoop. The architecture uses a master node dæmon called a Resource Manager consisting of two parts, a scheduler and Application Manager.

The scheduler is commonly called a pure scheduler in that it is only managing resource availability from the node manager on the data nodes. It also enforces scheduling policy as it is defined in the configuration files. The scheduler functions to schedule containers that are customizable collections of resources.

The Application Master is itself a container, albeit a special one, sometimes called container 0. The Application Master is responsible for launching subsequent containers as required by the job. The second part of the Resource Manager, called the Application Manager, receives job submissions and manages launching the Application Master. The Application Manager handles failures of the Application Master, while the Application Master handles failures of job containers. The Application Master then is really an application-specific container charged with management of containers running the actual tasks of the job.

Refactoring of resource management from the programming model of MapReduce makes Hadoop clusters more generic. Under

Figure 1.
YARN-Based Architecture of Hadoop 2



YARN, MapReduce is one type of available application running in a YARN container. Other types of applications now can be written generically to run on YARN including well-known applications like HBase, Storm and even MPI applications. The progress of MPI support can be seen in the Hamster project and a project called mpich2-yarn available on GitHub. YARN then moves from being a scheduler to an operating system for the Hadoop supporting multiple applications on a distributed architecture.

Architecturally, HPC workload management has many similarities to Hadoop workload management. Depending on the HPC workload management technology used, there is a set of master nodes containing cluster-controlling dæmons for accepting and scheduling jobs. The master node(s) in many cases contains special configurations including sharing of important cluster data via networked storage to eliminate SPOF of master services. On the worker node side, there exists one or more dæmons running to accept jobs and

Table 1. Comparison of Scheduling Options

POLICY OR FEATURE	HPC	HADOOP
FIFO	Available	Available
Fair Share	Available	Available
Time-Based Policies	Available	Technology Gap
Preemption	Available	Available
Exclusive Placement	Available	Technology Gap
Custom Algorithms	Available	Available
SLA- or QoS-Based	Available	Technology Gap
Round-Robin	Available	Technology Gap
Static and Dynamic Resources	Available	Available
Node Labeling	Available	Coming Soon
Custom Resources	Available	Technology Gap

report resource availability to the master node dæmons. Technologies from HPC, like Platform LSF and PBS Professional as well as other open-source variants like SLURM and Torque, are commonly seen in HPC.

These technologies are much older than Hadoop, and in terms of scheduling policy, they are more mature. They tend to share some basic tenets of scheduling policy that the Hadoop community is either in the process of addressing or has already.

First-In First-Out Scheduling

Many times this is the default policy used when a workload manager is first installed. As the name suggests, FIFO operates like a line or queue at a movie theatre.

Fair Share

Fair Share is a scheduling policy that attempts to allocate cluster resources fairly to jobs based upon a fixed number of shares per user or group. Fair share is implemented differently based upon the exact cluster resource management software used, but most systems have the concept of ordering jobs to be run in an attempt to even out the use of resources for all users. The specific ordering can be based upon a fixed number of shares or a percentage capacity of resources along with policies for an individual queue or a hierarchy of queues.

Time-Based Policies

Time-based policies come in a few different varieties. Queue-level time-based policies might be used to alter the configuration of a queue based upon time of day including allowing jobs to

Exclusive placement is important when users want to ensure that there is absolutely no contention for resources with other jobs within the selected nodes.

be submitted (enqueued) but not dispatched to nodes. Time-based policies enable concepts like using a cluster for a specific workload during business hours and an alternate workload overnight. Other time-based policies include dedicating the entire cluster or portion of a cluster for a specific use for a length of time. Additionally, draining a cluster of submitted jobs for maintenance windows is common.

Preemption

Preemption is the idea that some jobs can take the place of others that are currently running. Preemption is usually based upon the priority level of the job itself. The preempted job may be simply killed, suspended or possibly just requeued. All of these options come with benefits and disadvantages. Preemption in general tends to cause many internal political challenges but none as much as preemption by killing. Setting submitted high-priority work simply to be the next job to run when resources become available tends to balance the needs of high-priority work without the disruption potentially caused by a kill-style preemption model. An additional alternative would be to automate job requeue of preempted jobs instead of killing them. The best way to do preemption is intimately related to the workload profile.

Exclusive Job Placement

Exclusively placing jobs onto a node is an important job placement policy. Exclusively placing a job on a node means that no subsequent job could be placed on a node once a job is assigned to it. Exclusive placement is important when users want to ensure that there is absolutely no contention for resources with other jobs within the selected nodes. Users might request this type of placement when rendering video or graphics where memory is the rate-limiting factor in total wall time.

Exclusive placement can be enabled on most systems by matching the job resource request to encompass an entire single node. To do this, submitting users have to know specific hardware details of nodes in the cluster, and this approach also assumes node homogeneity. In many cases, users have no knowledge of the exact configuration of nodes, or there may be some level of heterogeneity across nodes in the cluster. Using a resource manager with a language for job submission that includes a client resource request flag to allow exclusive placement of jobs is highly desirable.

Custom Algorithms

Advanced cluster users eventually find that creating their own algorithm for custom job placement becomes required. In practice, these algorithms tend to be highly secret and bound to some proprietary process specific to the owner's vertical line of business. An example of a custom algorithm might include assigning specific jobs an immediate high priority based upon an organizational goal or specific project.

SLA- or QoS-Based Policies

Many times it is difficult to guarantee a job will complete within a required window. Most workload management systems have a direct or indirect way to configure scheduling policy such that jobs are guaranteed to finish within given time constraints. Alternatively, there may be ways to define custom qualities used to build scheduling policy.

Round-Robin Placement

Round-robin placement will take jobs from each queue in a specific order, usually within a single scheduling cycle. The queues are ordered by priority in most systems, but the exact behavior can be tricky depending upon the additional options used (for example, strict ordering in PBS Professional).

HPC Workload Manager Resource Types

Workload managers use resource request languages to help the scheduler place work on nodes. Many job placement scenarios include the specification of static or built-in resources as well as the ability to use custom-style resources defined using a script. Resource types tend to reflect programming primitives like boolean, numerical and string as well as properties like static and dynamic to reflect the nature of the values. Some of

these resource types are assigned specifically to hosts while others have to do with shared resources in a cluster like software licenses or allocation management (cluster use credits or chargebacks). These resources are all important in a multitenant distributed computing environment.

Hadoop Scheduling Policy

Hadoop currently makes use of mainly CPU and memory. There are additional selection criteria one can make when specifying container requests. The Application Master can specify a hostname, rack placement information and priority. Over time, Hadoop will benefit from a more mature resource specification design similar to HPC. One such use case would be a boolean host resource to specify placement of containers onto nodes with specific hardware (for example, a GPU). Even though very robust placement of containers can be accomplished in the Java code of the Application Master, resources requests probably need to be made more generic and available at a higher level (that is, during submission time via a common client). YARN allows for what it calls static resources from the submitting client and dynamic resources as those defined at runtime by the Application Master.

There are two built-in scheduling policies for Hadoop (excluding FIFO) at this time, but scheduling, like

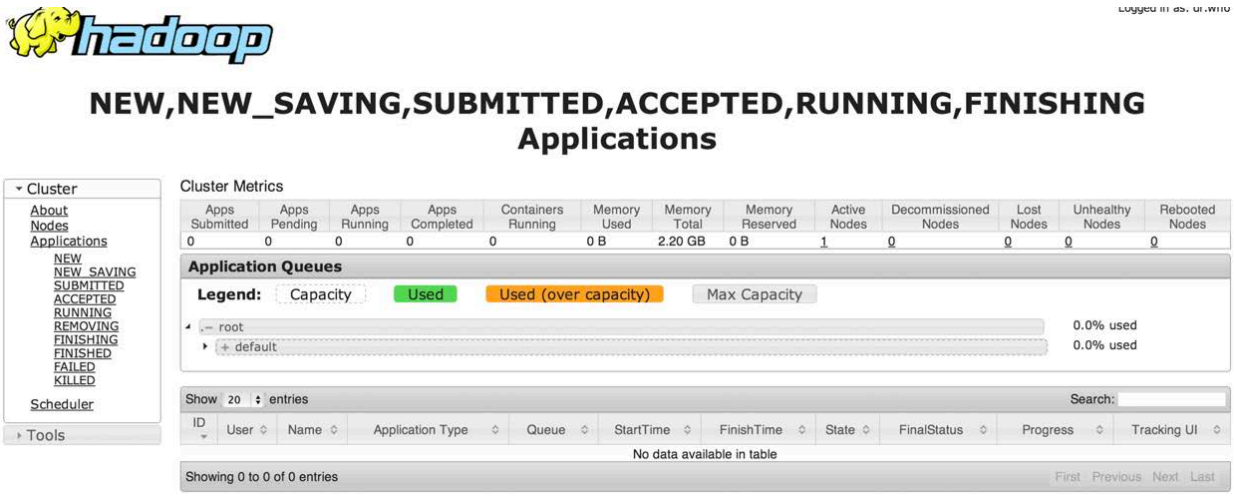


Figure 2. The scheduler page of the Resource Manager Web interface showing queue configuration and data on running applications.

most things in Hadoop, is pluggable. Setting `yarn.resourcemanager.scheduler.class` to the desired class in the configuration yarn-site.xml file can alter the specific scheduling type used. Custom scheduling policy classes can be defined here as well.

Scheduling policy for a Hadoop cluster is easy to access via a Web browser. Simply navigate to http://ResourceManager:port/cluster/scheduler using the Resource Manager hostname or IP and the correct port for the distribution of Hadoop being used.

FIFO

This is the standard first-in first-out method one might expect as a default scheduling option. It operates by accepting jobs and dispatching them in order received.

Capacity Scheduler

Hadoop's Capacity Scheduler was designed to provide minimum levels of resource availability to multiple users on the same cluster (aka multitenancy). Part of the power of Hadoop is having many nodes. The more worker nodes provided in a single cluster, the more resilient it is to failures. In large organizations with independent budgets, individual department heads might think it best to set up individual clusters to obtain resource isolation. Multitenancy can be accomplished logically using the Capacity Scheduler. The benefit of this design is not only better cluster utilization but also the improvement of system stability. Using more nodes decreases the importance of any one node in a node loss scenario by spreading out data as well as increasing cluster data and compute capacity.

The Capacity Scheduler functions through a series of queues. This includes hierarchical queues each with properties associated to direct the sharing of resources. The main resources include memory and CPU at this time. When writing an Application Master, the container requests can include resource requests, such as node resource (memory and CPU), a specific host name, a specific rack and a priority.

The capacity-scheduler.xml file contains the definition of queues and their properties. The settings in this file include capacity and percentage maximums along with total number of jobs allowed to be running at one time. In a multitenant environment, multiple child queues can be created below the root queue. Each queue

configuration contains a share of resources to be consumed by itself or shared with its children.

It's also common to see the use of access control lists for users of queues. Each queue in this case would receive a minimum capacity guaranteed by the scheduler. When other queues are below their capacity, another queue can use additional resources up to its configured maximum (hard limit).

Configurable preemption was added in Hadoop 2.1 for the capacity scheduler via ASF JIRA YARN-569. On the other hand, the complete isolation of resources so that no one job (AM or its containers) impedes the progress of another is accomplished in an operating-system-dependent way. Yes, Hadoop has matured so it will even run on Windows. For Linux, resource isolation is done via cgroups (control groups) and on Windows using job control. Future enhancements may even include the use of virtualization technologies, such as XEN and KVM, for resource isolation.

Fair Scheduler

The Fair Scheduler is another pluggable scheduling functionality for Hadoop under YARN. The Capacity and Fair Scheduler operate in a very similar manner although their nomenclature differs. Both systems schedule by memory and CPU; both systems use queues (previously called Pools) and attempt to provide a framework for sharing a common collection of resources. Fair Scheduler uses the concept of a minimum number of shares to enforce a minimum amount of resource availability with excess resources being shared with other queues. There are many similarities but a few nice unique features as well. Scheduling policy itself is customizable by queue and can include three options including FIFO, Fair Share (by memory) and a Dominant Resource Fairness (using both CPU and memory) that does its best to balance the needs of divergent workloads over time.

The yarn-site.xml file can include a number of Fair Scheduler settings including the default queue. A unique setting includes an option to turn on preemption that was previously preemption by killing and now includes a work-saving preemption option. One of the most important options in yarn-site.xml includes the allocation file location. The allocation file details queues, resource allotments as well as a queue-specific scheduling algorithm in XML.

The YARN Scheduler Load Simulator is a convenient tool for investigating options for scheduling via the options available to Hadoop.

YARN Scheduler Load Simulator

How should one choose between the two main options available? More important, how are the configurations tuned for optimal performance? The YARN Scheduler Load Simulator is a convenient tool for investigating options for scheduling via the options available to Hadoop. The simulator works with a real Resource Manager but simulates the Node Manager and Application Masters so that a fully distributed cluster is not required to analyze scheduling policy. One of the new configuration best practices should be possible to include time for scheduler tuning when initially setting up a new Hadoop cluster. This can be followed by analysis of scheduling policy at an interval going forward for continued optimization. Regardless of what type of scheduling is selected or how it is configured, there now is a tool to help each group determine what is best for its needs.

Scheduler simulation is a very technical field of study and something commercial HPC workload offerings have desperately needed for years. It is exciting to see a concrete method for analysis of Hadoop workloads, especially considering the effect a small change can make in throughput and utilization on a distributed system.

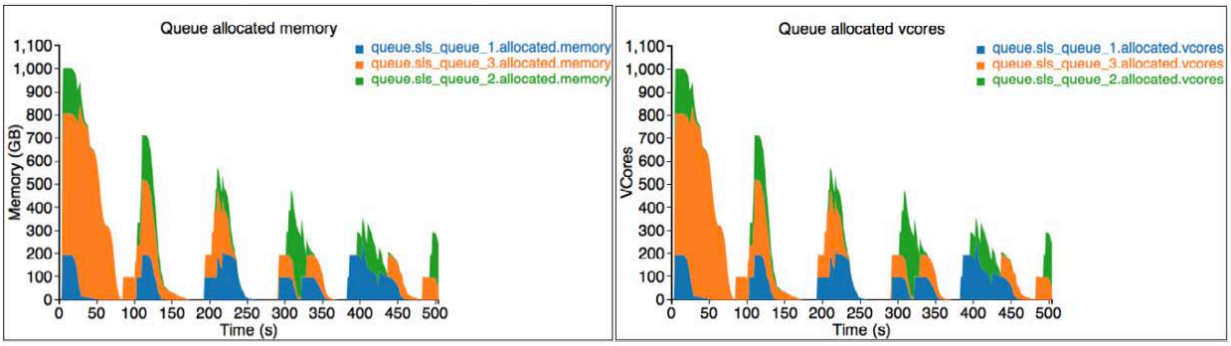


Figure 3. YARN Scheduler Simulator output showing memory and vcores for a queue.

Conclusions

Hadoop workload scheduling, much like the rest of Hadoop, is growing by leaps and bounds. With each release, more resource types and scheduling features become available, and it is exciting to see the convergence of Internet-scale distributed computing with the field of HPC that has been available for many years. One might argue some features from HPC workload management are needed in Hadoop. Examples, such as SLA-based scheduling and time-based policies are important operational examples of policies administrators expect. From a resource perspective, additional resource types also are needed.

The pace at which the open-source model innovates surely will close the gaps very soon. The participation of multiple groups and contributors in a meritocracy-based system drives not only the pace of innovation but quality as well.■

Adam Diaz is a longtime Linux geek and fan of distributed/parallel systems. Adam cut his teeth working for companies like Platform Computing, Altair Engineering and a handful of startups. His current endeavor is with Hortonworks helping companies make use of Hadoop. He can be reached at http://www.techtonka.com.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

Resources

Original YARN JIRA: **https://issues.apache.org/jira/browse/MAPREDUCE-279**

Hamster Project: **https://issues.apache.org/jira/browse/MAPREDUCE-2911**

mpich2-yarn: **https://github.com/clarkyzl/mpich2-yarn**

Apache Capacity Scheduler Site: **http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html**

Capacity Scheduler Preemption: **https://issues.apache.org/jira/browse/YARN-569**

Apache Fair Scheduler Site: **http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/FairScheduler.html**

Work-Saving Preemption: **https://issues.apache.org/jira/browse/YARN-568**

YARN Scheduler Load Simulator: **https://issues.apache.org/jira/browse/YARN-1021**

YARN Scheduler Load Simulator Demo: **http://youtu.be/6thLi8q0qLE**