Limits Configuration

Limits Overview

The Tractor *Limits* system can be used to monitor and enforce farm-wide restrictions on certain limited resources. It does this by simply maintaining counts associated with abstract site-chosen keywords. These are incremented as commands are launched and decremented when they exit. Sites can specify that new commands carrying particular limit keywords should only be launched when the current count is below a given threshold.

Consider an example where we have an application named "AwesomeUnicorn" that has been installed on all 100 of our renderfarm nodes. However, we only have 15 license seats for it, so we need to be careful to avoid launching more than 15 concurrent instances of it. Once 15 are running, we have to wait for some to exit before launching more, up to the limit of 15. Other types of commands from the job queue can run on the remaining idle farm nodes. Tractor maintains a tally of application names as they are launched, as well as other arbitrary *limit tag* names that the job script may attach to each command. This type of counting is called a SiteMax **global limit** because it applies to command keywords from any job, running anywhere on the farm.

The basic limit counting policy is controlled by the limit.config setting SiteMaxCounting. It has two possible settings today, in the default**"perInvocation"** mo de limit counts are incremented on each command dispatch. There is an alternate global counting scheme named **"perHost"** that increments counts only when the *first* command carrying a particular tag lands on a given farm host. Subsequent similar commands running on that same host will not increment that limit's count. The idea is to support tracking certain application licensing scenarios in which a license will support unlimited instances of an application running on the same host.

Tractor also maintains a BladeMax tally for each limit, indexed by **blade hostname.** This value can be used to limit the number of concurrent invocations that can run on any individual blade. For example, we might determine that the AwesomeUnicorn app is so awesome (or cpu intensive) that only one instance should ever run on a given blade at one time. Per-blade exceptions can also be specified.

Similarly, each limit also maintains a separate tally indexed by **job owner** name for each limit tag. This OwnerMax value can be used to limit individual users to a particular number of concurrent invocations that consume a given limit tag. For example, we might want to specify that no user should be using more than 5 AwesomeUnicorn licenses concurrently. As shown below, a site can specify a general owner limit and also several exceptions for particular users.

A limit definition can also specify a JobMax value that constrains each **individual job** on the queue to the given number of concurrent launches that consume that particular limit tag. It is a site-wide setting, affecting all jobs that use that limit.

Note: In addition to the JobMax value in limits.config definitions, *each job itself also has a* **maxActive** *attribute* that provides a different kind of per-job control over a job's "footprint" on the farm. Each job's maxActive attribute specifies a simple cap on the number of simultaneously active tasks from that job. It is a generic active task count restriction, independent of limit policies or command type. By default, each job's maxActive attribute is set to zero, meaning "unconstrained", and the usual site-wide Limit policies described on this page are used to control resource usage. The maxActive attribute is an additional constraint, to further reduce the job's blade usage.

Adaptive Farm Allocations were introduced in Tractor 2.0. This limit sharing mechanism provides a way to assign flexible percentages of a given site-wide limit to different types of jobs. So for example, jobs from project "X" get 10% of the farm while jobs from show "Y" get 60% and other jobs can take whatever remains. This type of sharing is described in more detail below.

The limits.config File

The limits.config file specifies a dictionary of limit definitions, in JSON notation. Each limit definition consists of required global and owner maximum concurrent counts, and optional modifiers. This file must be created in the site override directory.

A value of **-1** for SiteMax, OwnerMax, or BladeMax specifies that an **unlimited** number of invocations are allowed.

```
{
   "Limits":
   {
      "prman":    {"SiteMax": 100, "OwnerMax": -1},

      "thingy"   {  "SiteMax": 25,
                    "SiteMaxCounting": "perHost",
                    "OwnerMax": 5,
                    "OwnerExceptions": {"bob": 2, "alice": 10},
                    "BladeMax": 2,
                    "BladeExceptions": {"gigantor": 8, "myphone": 0}
                 },

      "aparatus": {"SiteMax": -1, "OwnerMax": 2}
   }
}
```

Specifying Limit Tags in Job Scripts

Tractor jobs can explicitly declare limit tags that they will consume. Typically they are specified with the -tags list parameter to Job or RemoteCmd. When specified at the job level the tags are used to extend each command's own list, if any.

```
RemoteCmd "/bin/AwesomeUnicorn mybest.corn" -tags "w00t spoon" -service "AUserver"
```

This example will increment limit tags named "AwesomeUnicorn", "w00t", and "spoon".

Blade Triggered Limit Tags

Sometimes it is desirable for a limit counter to be incremented whenever a certain type of blade is used, independent of the type of job that is running there. This can arise in situations where an administrator simply wants to know how many blades are in use from a given profile. However, this feature is typically used in the context of "limit sharing" allocations, described below.

To cause a blade assignment to trigger a limit increment, add a TaggedBlades value to the limit definition, it takes a list of profile names or host names that should trigger the counting.

Adaptive Farm Allocations

Limit Sharing provides a way to assign flexible percentages of a given limit to different types of jobs.

By default, a limit count is a simple "global" tally: the limit tags associated with any launched command are simply incremented on launch and decremented on exit. If a particular limit tally has reached its maximum allowed count, then commands using that tag are not allowed to launch until some prior commands exit.

The limit sharing variation causes these limit counts to be *attributed* to a particular named **project** and the system enforces a usage ratio between those projects.

So the required ingredients for sharing include:

- Add the **Job -projects** attribute specifying each job's affiliation. Unaffiliated jobs are attributed to the project named "default".
- Add the **Shares** dictionary to a limit definition in limits.config to describe that limit's allocations by project.
- Ensure that the shared limit is in effect during dispatching by either adding its **limit tag** name to job scripts, or by adding the *TaggedBlades*keywor d to the limit definition itself so that all assignments to the named blades (profiles) are tallied.

Jobs that do not specify a *project* affiliation are attributed to a project named *"default"*. Project affiliation can be added or changed in the Dashboard. The "Shares" named in a limit definition represent 100% of that limit usage. If the fractional named allocations do not sum to 1.0, then all remaining capacity is automatically allocated to the "default" project. A "Shares" definition can also explicitly define the size of the "default" allocation. If the sum of all allocations in a given limit definition is over or under 1.0 then the entire set of values is scaled proportionally so that the new total is 1.0.

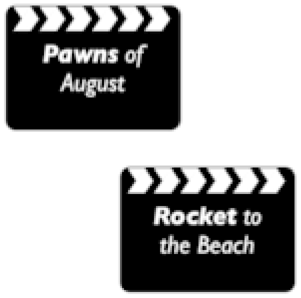
A Sharing Example

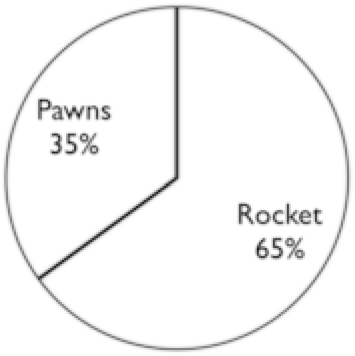
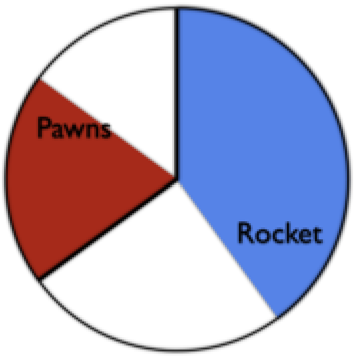
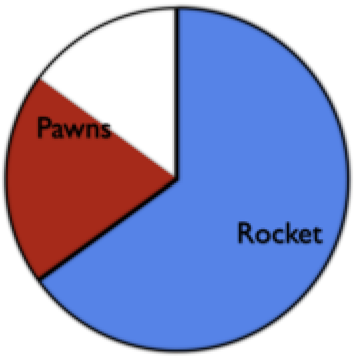
Somewhere in limits.config ...

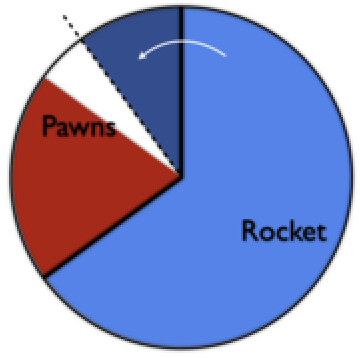
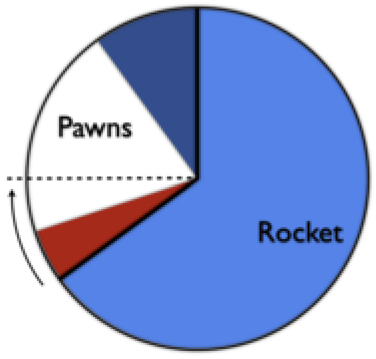
```
{
  "Limits":
  {
     "prman": {
         # a limit in the classical style
         "SiteMax": 100, "OwnerMax": -1
     },
     "theWholeFarm": {
        "SiteMax":  1000,
        "TaggedBlades": "Linux64", # for example, all blades in this Profile
        "Shares": {
           "Rocket": {"nominal": .65, "cap": .75},
           "Pawns":  {"nominal": .35, "reserve": 0.10}
        }
     },
     "someOtherLimitName": {
        "SiteMax":  234,
        "Shares": {
           "lookdev": {"nominal": .5},
           "research": {"nominal": .25, "cap": 0.5},
           "default": {"nominal": .25}
        }
     }
  }
}
```

Let's say we have two shows in production right now. We will spool our jobs with annotations giving each job's project affiliation(s).

```
Job -projects {Pawns}  -title "pawn_hair_test1" ...
or
Job -projects {Rocket} -title "rktpkt" ...
```



Referring to the example above, we created a limit tag name "theWholeFarm" that will abstractly represent all of the render farm blades using the "Linux64" blade profile.	theWholeFarm
Nominal Allocations -- Within that limit, we created two *Shares,* representing our allocation of theWholeFarm to each project. Allocations are given as fraction of the limit's SiteMax count. Rescaling occurs if the allocations sum to greater than 1.0.	
As tasks are assigned to blades, the tally for each project is incremented accordingly. The usage of theWholeFarm limit tag will always be the sum of counts in its shares.	
Nominal Maximum Utilization -- once the count for a project reaches that share's specified maximum, no additional tasks from that project will be dispatched until a previous one exits.	

Dynamic Overflow -- when one share is underutilized, others can optionally overflow their *nominal* alloc ation up to a larger predefined *cap* value.	
Reserved Allocations -- a share can optionally hold a reserved portion of the farm, some blades are kept idle until that project needs them.	 Pawns Rocket