Many VOs have adopted the pilot-based WMS paradigm, also known as "overlay infrastructure". In this type of environment, resources are organized into VO specific overlay pools, or "virtual clusters" (VC's). Each VO has full control over its own VC, and can thus easily implement policies better suited for their needs.

A broadly adopted pilot WMS is glideinWMS. One characteristic of glideinWMS is the clear separation between the VO handling the underlying infrastructure layer implementing the provisioning logic (VO Frontend) and the grid-interfacing layer responsible for the actual provisioning (Glidein Factory).

The Glidein Factory operations is mostly independent of the served VO, allowing for instances serving multiple VOs and thus reducing the total cost of ownership (TCO) through economies of scale.

The protocol between VO Frontend and Glidein Factory is based on the principle of constant pressure. When a VO Frontend needs a large number of additional resources, it does not ask for all of them in well defined chunks; instead, it asks for a stream of resource provisioning pilots, (with the understanding that the VO Frontend will be picking them in a random order).

A nice property of this approach is the possibility to request multiple streams from the same resource provider, e.g. the use of multiple Glidein Factories.

The most obvious benefit of having multiple Glidein Factories is providing redundancy to the glideinWMS ecosystem, thus eliminating the single point of failure. In case one Glidein Factory instance stops working, others pick up the load and the VOs hardly ever notice it.

In order to achieve full benefits from multiple Glidein Factories, each instance used by a VO must be configured in the same way. Full synchronization is however not possible.

We have developed a tool for selective cloning of attributes from one factory to another, allowing for semi-automated propagation of changes. (Only occasional manual adjustments needed)

A related benefit is scalability. By partitioning the glidein requests over multiple Glidein Factory instances the total number of supported provisioned resources scales linearly with the number of instances.

If we are to be 100% certain in our deployments, we should easily correlate the results in one instance with another. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operators are also responsible for monitoring the success rates of such requests, and act if too many are failing.

The main challenge is the sheer number of provisioning requests flowing through the system.

We have developed tools to filter out the log of well behaving glideins and sort the rest into the already broken ones. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operators are located at CERN, Fermilab, Indiana University and UCSD, and one operator is at KIT.

The increased head count also leads to the establishment of a collective memory. This allows for an easier handling of both personal needs of the various operators, such as vacation and sick days, as well as personnel turnover.

The overall experience has been very positive, and we look forward on continuing on this path.

The Glidein Factory operations is mostly independent of the served VO, allowing for instances serving multiple VOs and thus reducing the total cost of ownership (TCO) through economies of scale.

The protocol between VO Frontend and Glidein Factory is based on the principle of constant pressure. When a VO Frontend needs a large number of additional resources, it does not ask for all of them in well defined chunks; instead, it asks for a stream of resource provisioning pilots, (with the understanding that the VO Frontend will be picking them in a random order).

A nice property of this approach is the possibility to request multiple streams from the same resource provider, e.g. the use of multiple Glidein Factories.

The most obvious benefit of having multiple Glidein Factories is providing redundancy to the glideinWMS ecosystem, thus eliminating the single point of failure. In case one Glidein Factory instance stops working, others pick up the load and the VOs hardly ever notice it.

In order to achieve full benefits from multiple Glidein Factories, each instance used by a VO must be configured in the same way. Full synchronization is however not possible.

We have developed a tool for selective cloning of attributes from one factory to another, allowing for semi-automated propagation of changes. (Only occasional manual adjustments needed)

A related benefit is scalability. By partitioning the glidein requests over multiple Glidein Factory instances the total number of supported provisioned resources scales linearly with the number of instances.

If we are to be 100% certain in our deployments, we should easily correlate the results in one instance with another. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operators are also responsible for monitoring the success rates of such requests, and act if too many are failing.

The main challenge is the sheer number of provisioning requests flowing through the system.

We have developed tools to filter out the log of well behaving glideins and sort the rest into the already broken ones. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operations is mostly independent of the served VO, allowing for instances serving multiple VOs and thus reducing the total cost of ownership (TCO) through economies of scale.

The protocol between VO Frontend and Glidein Factory is based on the principle of constant pressure. When a VO Frontend needs a large number of additional resources, it does not ask for all of them in well defined chunks; instead, it asks for a stream of resource provisioning pilots, (with the understanding that the VO Frontend will be picking them in a random order).

A nice property of this approach is the possibility to request multiple streams from the same resource provider, e.g. the use of multiple Glidein Factories.

The most obvious benefit of having multiple Glidein Factories is providing redundancy to the glideinWMS ecosystem, thus eliminating the single point of failure. In case one Glidein Factory instance stops working, others pick up the load and the VOs hardly ever notice it.

In order to achieve full benefits from multiple Glidein Factories, each instance used by a VO must be configured in the same way. Full synchronization is however not possible.

We have developed a tool for selective cloning of attributes from one factory to another, allowing for semi-automated propagation of changes. (Only occasional manual adjustments needed)

A related benefit is scalability. By partitioning the glidein requests over multiple Glidein Factory instances the total number of supported provisioned resources scales linearly with the number of instances.

If we are to be 100% certain in our deployments, we should easily correlate the results in one instance with another. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operators are also responsible for monitoring the success rates of such requests, and act if too many are failing.

The main challenge is the sheer number of provisioning requests flowing through the system.

We have developed tools to filter out the log of well behaving glideins and sort the rest into the already broken ones. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operations is mostly independent of the served VO, allowing for instances serving multiple VOs and thus reducing the total cost of ownership (TCO) through economies of scale.

The protocol between VO Frontend and Glidein Factory is based on the principle of constant pressure. When a VO Frontend needs a large number of additional resources, it does not ask for all of them in well defined chunks; instead, it asks for a stream of resource provisioning pilots, (with the understanding that the VO Frontend will be picking them in a random order).

A nice property of this approach is the possibility to request multiple streams from the same resource provider, e.g. the use of multiple Glidein Factories.

The most obvious benefit of having multiple Glidein Factories is providing redundancy to the glideinWMS ecosystem, thus eliminating the single point of failure. In case one Glidein Factory instance stops working, others pick up the load and the VOs hardly ever notice it.

In order to achieve full benefits from multiple Glidein Factories, each instance used by a VO must be configured in the same way. Full synchronization is however not possible.

We have developed a tool for selective cloning of attributes from one factory to another, allowing for semi-automated propagation of changes. (Only occasional manual adjustments needed)

A related benefit is scalability. By partitioning the glidein requests over multiple Glidein Factory instances the total number of supported provisioned resources scales linearly with the number of instances.

If we are to be 100% certain in our deployments, we should easily correlate the results in one instance with another. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operators are also responsible for monitoring the success rates of such requests, and act if too many are failing.

The main challenge is the sheer number of provisioning requests flowing through the system.

We have developed tools to filter out the log of well behaving glideins and sort the rest into the already broken ones. However, this still leaves a substantial number of logs that require some human parsing.

The Glidein Factory operations is mostly independent of the served VO, allowing for instances serving multiple VOs and thus reducing the total cost of ownership (TCO) through economies of scale.

The protocol between VO Frontend and Glidein Factory is based on the principle of constant pressure. When a VO Frontend needs a large number of additional resources, it does not ask for all of them in well defined chunks; instead, it asks for a stream of resource provisioning pilots, (with the understanding that the VO Frontend will be picking them in a random order).

A nice property of this approach is the possibility to request multiple streams from the same resource provider, e.g. the use of multiple Glidein Factories.