Flexible Pilot Jobs Framework for Distributed High Throughput Computing

FERMILAB-TM-2895-STUDENT

This manuscript has been authored by Fermi Forward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Flexible Pilot Jobs Framework for Distributed High Throughput Computing

Franco Terranova

Under the supervision of

Marco Mambelli

Fermilab Italian Summer School

Fermi National Accelerator Laboratory
Scientific Computing Division

ABSTRACT

Experimental particle physics has been at the forefront of analyzing the world's largest datasets for decades.
The high-energy physics (HEP) community was among the first to develop suitable software and computing tools for this purpose.
GlideinWMS is a Glidein-based workload management system whose purpose is to provide experiments like CMS at CERN, DUNE at Fermilab, and others, a way to access and efficiently use vast amounts of computing resources.
This system wants to provide a simple way to submit jobs to a set of computing resources, that will be provided to users behind the scenes.
Glideins are the pilot jobs executed on the worker nodes at the grid sites, performing operations such as hardware detection, environment setup, and error handling.
After all these operations, they will launch the actual user job.
Many grid sites are supported, such as shared clusters, Google CE, and AWS.
My internship aimed to design and code a flexible pilot jobs framework that will replace the one used by GlideinWMS, developing a modular and flexible skeleton of the Glidein and adding further functionalities.
My project also focused on the application of machine learning techniques as support to this management system.

Contents

1 GlideinWMS **1**

1.1 GlideinWMS Architecture . . . 2

1.2 Glideins . . . 3

1.3 Worker nodes . . . 3

2 Flexible and modular Glidein skeleton **4**

2.1 glidein_startup.sh modularization . . . 5

2.2 Bats tests . . . 7

2.2.1 Timeout feature . . . 7

3 External files management **8**

3.1 Life-cycle phases of management . . . 9

3.2 Management priority . . . 10

3.3 Tarballs handling . . . 12

3.4 File types . . . 13

3.5 File transfer methods . . . 13

3.6 File versions conversion . . . 14

4 Job/Site matching problem **15**

4.1 Dataset . . . 16

4.2 CPU time series analysis . . . 16

4.3 Memory time series analysis . . . 18

4.4 Failure Prediction . . . 18

4.4.1 Logistic Regression Model . . . 20

4.5 Score calculation . . . 21

Appendix A: AI techniques for the job/site matching problem . . . 25

Chapter 1

GlideinWMS

Grid computing is becoming very useful and popular for the scientific community with high computing demands, like high energy physics (HEP).
High throughput computing (HTC) aims to provide an efficient and effective scheduling of user jobs on top of computing resources.
Computing resources are distributed over many independent sites, and typically a middleware layer is used for enabling communication and data management among them. Distributed HTC aims to aggregate many unrelated HTC systems.
GlideinWMS is a Glidein-based workload management system whose purpose is to provide a simple way to access the computing resources for the HEP community.
This system works on top of HTCondor and its main building blocks are Glideins, a mechanism by which one or more remote resources temporarily join a local HTCondor pool.

1.1 GlideinWMS Architecture

The GlideinWMS architecture is composed of different services, interacting with each other as visible in Figure 1.1.

- Users submit jobs to the User Pool (HTCondor schedd process).
- The GlideinWMS Frontend polls the user pool to make sure that there are enough workers to satisfy user jobs.
  It then submits requests to the GlideinWMS Factory asking for the submission of Glideins.
- The GlideinWMS Factory receives requests from the Frontend(s) and other clients and submits the Glideins to the grid sites.
- The computing resources receive the Glideins and start an HTCondor startd process that joins the User Pool.
- The user jobs run on the newly added startds.

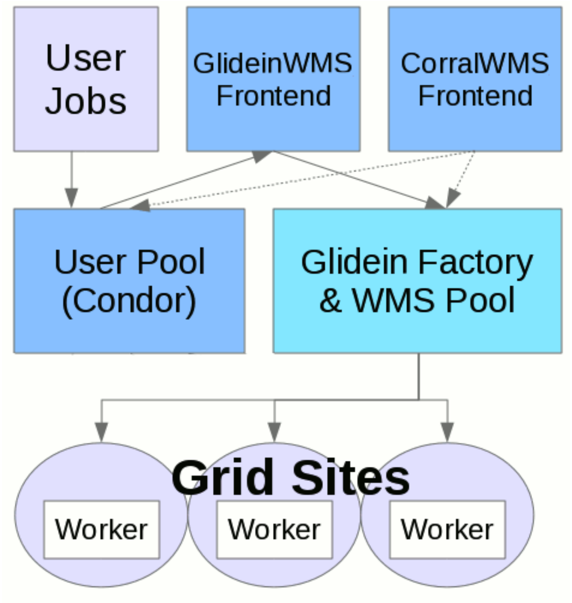


Figure 1.1: GlideinWMS Architecture

The result is that users can submit regular HTCondor jobs to the local queue and computing resources will be provided by the Glidein Factory behind the scenes.
All the burdens typically managed by the user, in terms of queues, grid entry points, and provisioning of worker nodes are handled by GlideinWMS.
From the final user point of view, the user pool will just magically grow and shrink as needed.

1.2 Glideins

A Glidein is a pilot job submitted on the worker node that acquires and prepares the resources where the job will run, with a particular focus on hardware detection, environment setup, and error handling.
The Glidein will finally start the job and monitor its execution.
The Glidein is requested by the Frontend, launched by the Factory on the worker nodes, and once launched, it will join the virtual cluster and start accepting user jobs.

1.3 Worker nodes

A worker node is a machine managed by a resource manager characterized by its resources (CPU, RAM, Disk, etc.).
Virtualization techniques are used in order to create the abstraction of virtual machines on top of the physical ones. These techniques ensure flexibility, scalability, and safety, while creating significant cost savings.
Worker nodes used by GlideinWMS come from different sources, such as the Fermilab Grid Computing Center, AWS, GoogleCE, and the Open Science Grid.

Chapter 2

Flexible and modular Glidein skeleton

The Glidein structure is described by a single Bash script, whose purpose is to handle the overall lifecycle of the Glidein.
Modular programming is a software design technique that emphasizes the separation of functionalities of a program into independent and interchangeable modules.
Its goal is to split the code into separate parts, modules, defining their boundaries, API, and minimizing the connections between elements in different modules.
Modularization techniques have been applied to the Glidein structure in order to go towards a more flexible and modular structure.
These techniques make development quicker and easier, as smaller subprograms are easier to understand, write, and design than larger ones.

2.1 glidein_startup.sh modularization

glidein_startup.sh is a Bash script describing the workflow of the Glidein.
It is a long shell script that performs the following operations:

- Downloads other scripts and binaries using HTTP
- Validates the node
- Prepares the environment, also installing user software if needed
- Configures Condor daemons (policies, security, proxies, etc.)
- The condor_startd process is then launched

After the modularization of the script, a new structure has been highlighted, as shown in the following structure tree.
The file's name will provide information about the specific functionalities handled by the script's functions.

```
| glidein_startup.sh
|   |__ do_start_all
|   |__ spawn_multiple_glideins
|   |__ setup_OSG_Globus
|   |__ check_file_signature
|   |__ parse_arguments
|   |__ prepare_workdir
|   |__ create_glidein_config
|   |__ get_data
|   |__ source_data
|   |__ extract_and_source_all_data
|   |__ _main
|__ glidein_cleanup.sh
|   |__ glidein_cleanup
|   |__ early_glidein_failure
|   |__ glidein_exit
|__ utils_crypto.sh
|   |__ md5wrapper
|   |__ set_proxy_fullpath
|__ utils_gs_filesystem
    |__ dir_id
```

```
 |
 |__copy_all
 |__add_to_path
 |__automatic_work_dir
|utils_gs_http
 |__get_repository_url
 |__add_periodic_script
 |__fetch_file_regular
 |__fetch_file
 |__fetch_file_try
 |__fetch_file_base
 |__perform_wget
 |__perform_curl
|__utils_gs_log
 |__print_tail
 |__usage
 |__parse_options
|__utils_gs_tarballs.sh
 |__fixup_condor_dir
 |__get_untar_subdir
|__utils_log.sh
 |__log_warn
 |__log_debug
 |__print_header_line
|__utils_params.sh
 |__params_get_simple
 |__params_decode
 |__params2file
|__utils_signals.sh
 |__signal_trap_with_arg
 |__signal_on_die
 |__signal_on_die_multi
 |__signal_ignore
 |__signal_add_child
 |__signal_set_children
|__utils_xml.sh
 |__construct_xml
 |__extract_parent_fname
 |__extract_parent_xml_detail
 |__basexml2simplexml
 |__simplexml2longxml
 |__create_xml
```

2.2 Bats tests

Bats is a TAP-compliant testing framework for Bash.
It provides a simple way to verify that the UNIX programs behave as expected using unit tests.
A Bats test file is a Bash script with special syntax for defining test cases.
Starting from the structure derived after modularization, a Bats test has been defined for each module.

2.2.1 Timeout feature

The runtest.sh script of GlideinWMS is a runner script for the different tests, among which also Bats tests.
A per-test timeout feature has been added to the script in order to avoid possible problems of non-responding tests that could interrupt the execution of subsequent tests.
A new timeout option that the user running the tests can set has been defined, and this option will avoid this issue.

```
-k TOUT
   sets a timeout of TOUT seconds for the execution of each test (BATS file).
   A TERM signal is sent if the test is still running after TOUT seconds and
   a hard kill (KILL signal) is sent 20 seconds after the previous signal if
   the test did not end yet.
```

Chapter 3

External files management

The experiments running jobs on GlideinWMS can upload external files of different types to help glidein_startup.sh to do its tests and support their jobs. These files need to be downloaded and managed during their execution.
The location, download/execution order, and attributes of these files need to be specified. The Glidein startup script will pull these files, validate them and execute any action requested (execute, untar, just keep them, ...).
Two XML configuration files will provide the information needed to manage these files, *frontend.xml* on the Frontend side and *GlideinWMS.xml* on the Factory side.
In these XML descriptors, a *files* section will contain all the information needed.

Listing 3.1: XML *files* section example

```
<files>
    <file absfname="filepath" relfname="filename" prefix="cron_prefix"
        executable="boolean" after_group="boolean" period="seconds(int)" />
    ..
</files>
```

My modifications to the external files management consisted of the following improvements:

- Execution of custom scripts during different parts of the life-cycle of the Glidein, not only the setup
- Revised coordination of the list of custom scripts inside a life-cycle phase
- Definition of new files' attributes
- Definition of new methods for transferring script files

3.1 Life-cycle phases of management

My first improvement focused on adding the possibility to deal with external files in different phases of the life-cycle of the Glidein.
The new overall life-cycle phases for the possible management of files are the following:

- startup
- pre_job
- after_job
- cleanup
- periodic:period

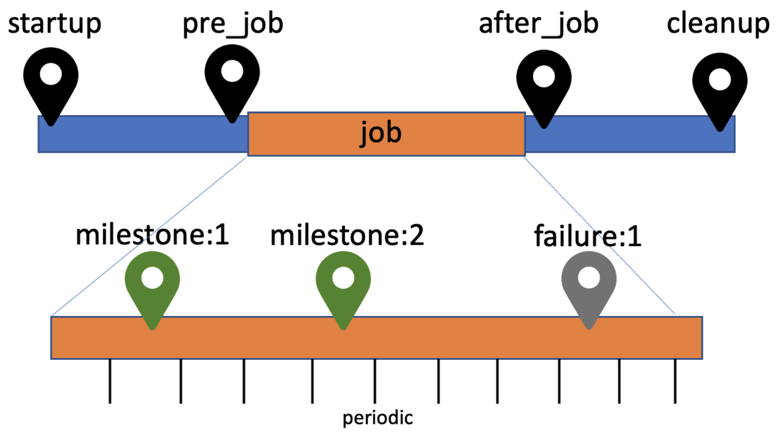


Figure 3.1: Glidein's life-cycle phases

Using the *time* attribute of the *file* tag, the user will be able to provide this information:

Listing 3.2: XML files section example

```
<files>
    <file .. time="time_phase1[, time_phase2, ..]" .. />
    ..
</files>
```

Multiple comma-separated values can be provided if the same file has to be managed in different life-cycle phases.
Two additional new possible phases have been suggested as ideas for future versions:

- failure:exit_code
- milestone:code

With the first option, the user can decide to manage an external file in front of a failure with a certain exit code.
With the second option, the user can set some milestones during the execution of the job (using some provided well-known functions) and ask for the management of external files in front of these milestones.

3.2 Management priority

Users may need to affect the management order of external files.
In the current version of GlideinWMS, the *after_group* and *after_entry* boolean flags in the file's attributes affect the files' handling order.
My modifications want to provide a simpler way to allow the user to alter this order, providing this possibility also inside a life-cycle phase of the Glidein.
With my modifications, the user will be able to set the priority attribute, specifying a priority value using a string code representation or an integer representation.

Listing 3.3: files' *priority* section example

```
<files>
    <file .. priority="e-g-" .. />
    <file .. priority="27" .. />
    ....
</files>
```

The following syntax meaning has been associated with the string code representation:

e: entry
g: group
+: after/post
-: before/pre
=: in that exact moment

Order	String code representation	Integer code representation
Factory pre_entry	e-[g-]	10
Frontend pre_entry pre_group	e-g-	20
Frontend pre_entry group	e-g=	30
Frontend pre_entry after_group	e-g+	40
Entry	e=[g-,g+,g=]	50
Frontend after_entry pre_group	e+g-	60
Frontend after_entry group	e+g=	70
Frontend after_entry after_group	e+g+	80
Factory after_entry	e+[g+]	90

Table 3.1: Files' priority ordering

The user can also choose to specify the priority value using the integer code representation with an integer value in the range [0, 99].
The predefined integer values are purposefully chosen as multiples of 10, so that intermediate values can be used to control in a more granular way the file handling order.

```
# File: file_list
#
# Version: 3.11.0
# Time       OrderedFileName   RealFileName    Type  Period  Prefix  Id
############################################################################
 startup    10_condor_vars.sh  condor_vars.sh  exec      ..
 startup    40_condor_file.sh  condor_file.sh  exec      ..
```

Listing 3.4: Example of file descriptor

A global file descriptor is used internally to handle the ordering of files considering both the life-cycle phase and the file priority.

3.3 Tarballs handling

The user can group files in a tarball and provide it to GlideinWMS, which is going to handle it properly.

Listing 3.5: XML tarball files section example

```
<files>
    <file type="untar:folder_name" cond_attr="cond_attr"
        absdir_outattr="attr_name" .. />
</files>
```

The new format allows the user to specify the destination folder as a qualifier (by default it will be the name of the tarball itself).
The user can also specify the types of files and all the information of files contained inside a tarball, for them to be handled properly.
These files will not be downloaded, since this will happen only for the tarball in which they are contained, but the proper values will be added in the file descriptor.

Listing 3.6: XML section example of files in tarballs

```
<files>
    <file .. type="executable" tar_source="tar_filename" .. />
    ..
</files>
```

3.4 File types

After the overall improvement of the management of external files, new file types have been defined.

- source files, including bash files to be sourced
- library:type, including library types to be used (e.g. library:shell)

New possible ideas about file types have been proposed.

- web files, to create custom web dashboards for different purposes, such as monitoring the execution of the job.
  The *web_group* attribute can be used to let the user provide web files of different web dashboards.
  The Glidein would have to start a Web server and serve these pages.

  Listing 3.7: XML web files example

  ```
  <file .. type="web" web_group="group_name" .. />
  ```

- container:type, to let the user specify the information of custom containers that will be launched during the execution of the Glidein. Note that the Glidein has already the option to run a script in a users container. This additional type would be to support custom containers, e.g. to run services.

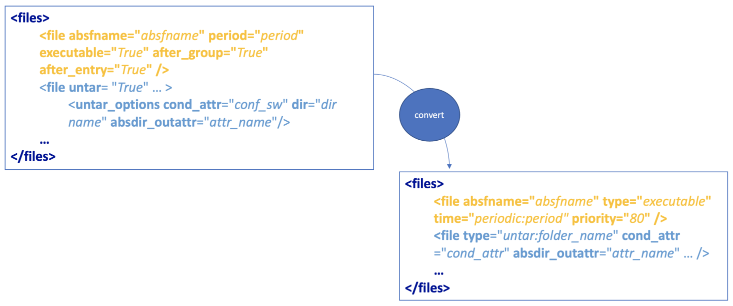
3.5 File transfer methods

New possible methods have been proposed for transferring files:

- Git repository
- URL (FTP, Cloud Storage)
- Database access information

3.6 File versions conversion

We have to plan for upgrades and compatibility since both some internal formats used by the Glidein and the configuration files changed. The GlideinWMS systems have many Factories and Frontends talking together and providing different Glidein components. It is unreasonable to expect all to upgrade at the same time. But it is reasonable to ask to upgrade the Factories first. The main Glidein components come from the Factory and we'll be able to operate also with older Frontends if the new Glideins can handle different versions of XML descriptor files.
Factory and Frontend operators have sometime long and complex XML configuration files. We want to reduce their effort caused by upgrades. The operators will be requested to convert the format to the new version when upgrading.
A converter script has been defined to let the user move to the new format.



```
<files>
      <file absfname="absfname" period="period"
      executable="True" after_group="True"
      after_entry="True" />
      <file untar= "True" ... >
            <untar_options cond_attr="conf_sw" dir="dir
            name" absdir_outattr="attr_name"/>
      ...
</files>
```

```
<files>
      <file absfname="absfname" type="executable"
      time="periodic:period" priority="80" />
      <file type="untar:folder_name" cond_attr
      ="cond_attr" absdir_outattr="attr_name" ... />
      ...
</files>
```

Figure 3.2: File versions converter

Chapter 4

Job/Site matching problem

What if we can use artificial intelligence to predict what is the best site where the Glidein should be spawned?
The Frontend, in fact, will have to choose toward which site the Glidein should be launched.
Currently it will request Glideins on all sites that state to provide enough resources.
We aim to allocate the Glidein to the site that provides the largest amount of resources while minimizing the probability of failure.
Some sites give information about the number of resources that are going to provide for the job, while some others don't.
The workflow we want to follow to solve the job/site matching problem is the following:

- Predict the amount of CPU and Memory that is going to be provided by each site: *CPUProvided*, *MemoryProvided*
- Consider only the sites for which the resources provided are enough for our job, which means:
  $CPURequested \leq CPUProvided$
  $MemoryRequested \leq MemoryProvided$
- Calculate the probability of failure of each site: $P_{failure}$
- Calculate a cumulative score that allows us to take this decision

More details about the analysis can be found in Appendix A.

4.1 Dataset

To perform our analysis, we considered the dataset *hepcloud-classads-slots*, available through Kibana, containing different information about classads.

4.2 CPU time series analysis

A time series analysis was performed comparing different forecasting methods to predict the average total hourly CPU that a certain site could provide.
The analysis has been performed on the overall data, regardless of the site, to train a general model that could generalize the site-by-site average total hourly CPU forecasting.

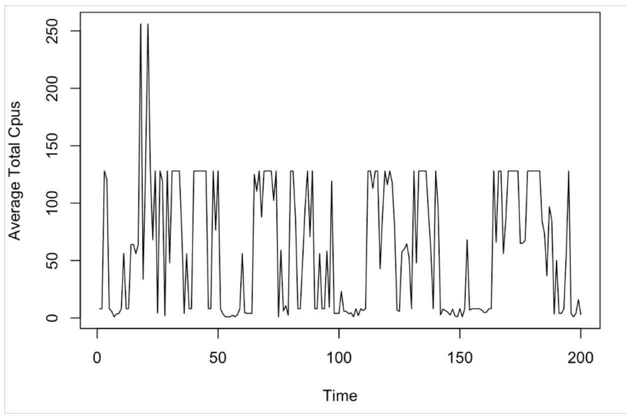


Figure 4.1: Average hourly total CPU time series

Exploring the time series with the auto-correlation function, a possible seasonality with a period of 24 hours was highlighted.
Going towards the decomposition of the time series, the additive decomposition and multiplicative decomposition were compared, choosing the one that was able to follow more accurately the nature of the series, separating seasonality, from trend, and noise.

$$\textit{Additive decomposition: } X_i = T_i + S_i + E_i$$

$$\textit{Multiplicative decomposition: } X_i = T_i * S_i * E_i$$

By analyzing the residues of the two decomposition methods, the additive decomposition seemed to provide better results.
Different forecasting methods have been compared:

- Holt-Winters method with additive decomposition
- Manually-created regression model for time series
- Yule-Walker method
- Least squares method

Among the different models, the Holt-Winters method and the Least squares method seemed to achieve better results.
A cross-validation comparison and a residues analysis have been performed to determine the best method to filter our time series and forecast the next values.
After this comparison, the Least squares method was chosen as the forecasting method for the next values of the time series.

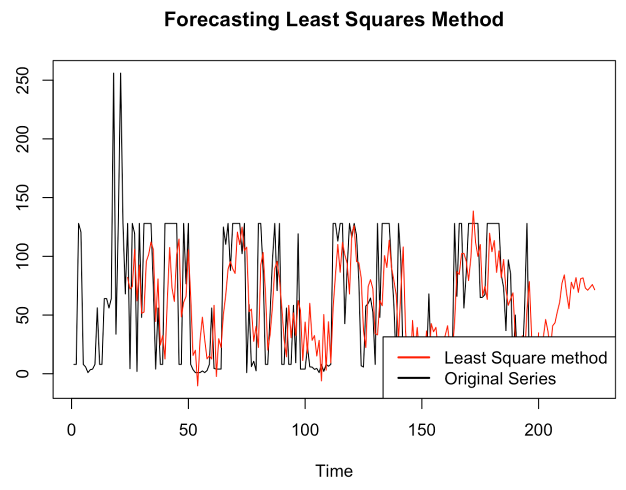


Figure 4.2: Time series filtering and forecasting

4.3 Memory time series analysis

The same approach was used for the time series analysis of the data regarding the memory allocation, where the Holt-Winters method seemed to provide the best results, with an additive decomposition and the following parameters:

Parameter	Value
α	0.30
β	0.03
γ	0.70
Initial intercept	11352
Initial slope	10580

Table 4.1: Parameter choices for the Holt-Winters method

A regressive model was used to obtain the initial values of intercept and slope.

4.4 Failure Prediction

As previously mentioned, we also want to find a way to minimize the probability of failure when we allocate a Glidein to a node.
Different probabilistic classification methods were compared to predict the failure of a node in a certain site:

- Linear regression for classification
- Logistic regression
- Linear discriminant analysis
- Quadratic discriminant analysis

Past data regarding failures and non-failures of the Glidein has been analyzed to train our supervised algorithm.
Among the features considered to train our classifiers, factors regarding the amount of resources of the worker nodes and information about the time needed for the job to be executed have been used.
During the training phase, all algorithms seemed to perform moderately and similarly.
A K-Fold cross-validation and a site-by-site cross-validation were faced to determine

the best classifier, using as metrics the accuracy's mean and standard deviation, and the ROC curve.
In front of this comparison, it emerged that the linear regression method for classification performed quite badly, while other methods seemed to have similar metrics' values.
A robustness analysis was performed by introducing wrong information to determine how robust each model was.
Data was altered by flipping at each iteration the class value of a sample.
The model was trained on the altered data and its accuracy was tested on real data.

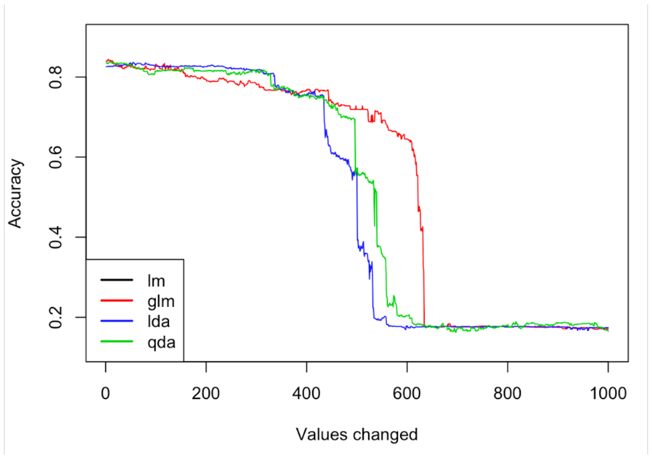


Figure 4.3: Models' robustness comparison

The logistic regression method has shown a higher degree of robustness, needing a higher number of values changed before losing a large amount of its accuracy.

4.4.1 Logistic Regression Model

A choice of a different threshold value than the default one provided by the model was performed to improve the prediction of failures and achieve a good trade-off between accuracy and sensitivity.

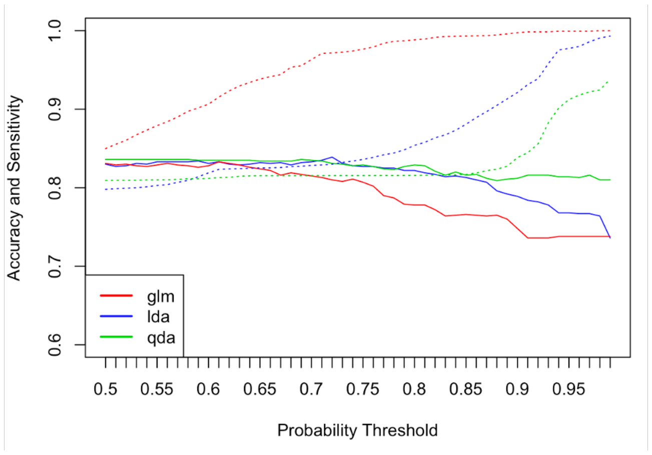


Figure 4.4: Accuracy/Sensitivity comparison

A probability threshold equal to 0.68 has been chosen, allowing the model to improve decisively its sensitivity while losing a few percentages of accuracy.
The following results were achieved with the logistic regression.

Accuracy	Sensitivity
0.81	0.93

Prediction outcome

		p	**n**	**total**
actual value	**p′**	6554	486	7040
	n′	3626	15117	18743
	total	10180	15603	

Table 4.2: Accuracy, sensitivity and confusion matrix

4.5 Score calculation

A *resources_score* taking into consideration the number of resources forecasted has been defined as follows:

$$resources\_score = \frac{CPU - \min(CPUs)}{\max(CPUs) - min(CPUs)} \times 50 + \frac{Memory - \min(Memories)}{\max(Memories) - min(Memories)} \times 50$$

The CPU and memory values forecasted with the corresponding methods are scaled in the range [0, 50] to give them the same weight.
The *resources_score* will be a value ranging from [0, 100].
The overall *score* taking into consideration also the probability of failure will allow this probability to decrease the *resources_score* depending on its value.

$$score = resources\_score \times (1 - P_{failure})$$

The site with the highest score will then be chosen for the allocation of the Glidein.

CONCLUSIONS

An approach to solve the problem of the allocation of grid resources is to create a homogeneous virtual private pool of computing resources and use a standard batch system to manage them.
In order to gather resources, batch system components are packaged as pilot jobs and sent to the Grid pools.
My project at Fermilab mainly regarded the modification of the pilot jobs structure, providing a more elastic and personalizable skeleton, and the improvement its features.
The following achievements were accomplished:

- Designed a flexible, modular, and customizable structure of the Glidein
- Added unit tests
- Redesigned the custom script management
- Applied AI techniques for the job/site matching problem

The modifications made to the design of the Glidein will be included in release 3.11.0 of GlideinWMS.
The other improvements will be considered in further releases.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere and deep gratitude to my supervisor, **Marco Mambelli**, working on the Scientific Computing Division, for his kind and constant support during my internship project. It has been an absolute privilege to work with him.
His valuable advice, critical criticism and active supervision encouraged me to sharpen my research methodology and was instrumental in shaping my professional outlook.

I also want to express my gratitude towards **Prof. Simone Donati** and **Prof. Giorgio Bellettini**, for providing such a wonderful environment filled with continuous encouragement and support. I would also like to thank the entire group behind the Fermilab Italian Summer School and the Fermilab's Scientific Computing Division for their constant encouragement and assistance they have provided me.

References

[1] GlideinWMS project, available at https://github.com/glideinWMS/glideinwms

[2] GitHub PR for the Glidein modularization, available at https://github.com/mambelli/glideinwms/pull/12

GitHub PR for the external files management modifications, available at https://github.com/glideinWMS/glideinwms/pull/210

[3] Job/site intelligent allocation, GitHub, available at https://github.com/glideinWMS/contrib/tree/main/AIforJobSiteAllocation

[4] Marco Mambelli, "GlideinWMS Overview" presentation, available at https://glideinwms.fnal.gov/presentations/intro/GlideinWMS.pdf

[5] GlideinWMS website, available at https://glideinwms.fnal.gov/doc.prd/index.html

[6] GlideinWMS API Documentation, available at https://glideinwms.fnal.gov/api/

[7] Development workflow, available at https://github.com/glideinWMS/glideinwms/wiki/Development-Workflow

[8] Igor Sfiligoi, "Structural overview of the GlideinWMS," *Fermilab, (2008).*

Appendix A: AI techniques for the job/site matching problem

Artificial intelligence can be a key instrument for the prediction of the best allocation of jobs to grid sites.
We want to allocate the Glidein to the site that provides the largest amount of resources while minimizing the probability of failure.
In this appendix, a more detailed comparison of time series forecasting methods of CPU and Memory, and more details about the classifiers' comparison are reported.

A.1 CPU Time Series Analysis

The autocorrelation function of the time series has been explored to discover a possible seasonality in the time series.

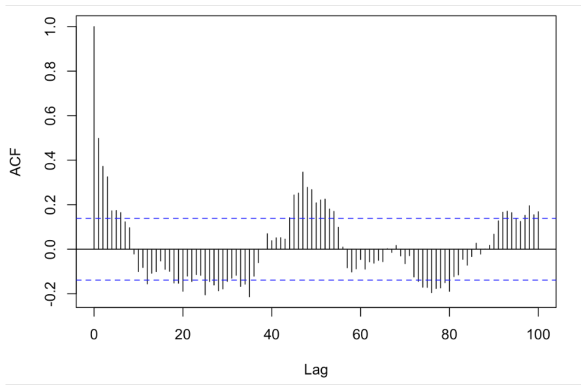


Figure 4.5: Auto-correlation function

Seasonality with a period of 24 hours was suggested by the analysis.
Highlighted the possible period of the time series, its decomposition has been performed.

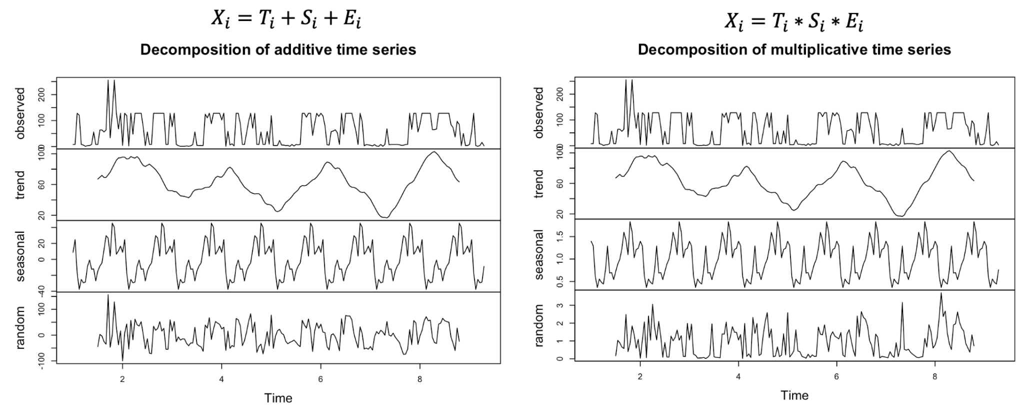


Figure 4.6: Decompositions comparison

To determine the best decomposition structure, the residues of the two models were analyzed and compared.

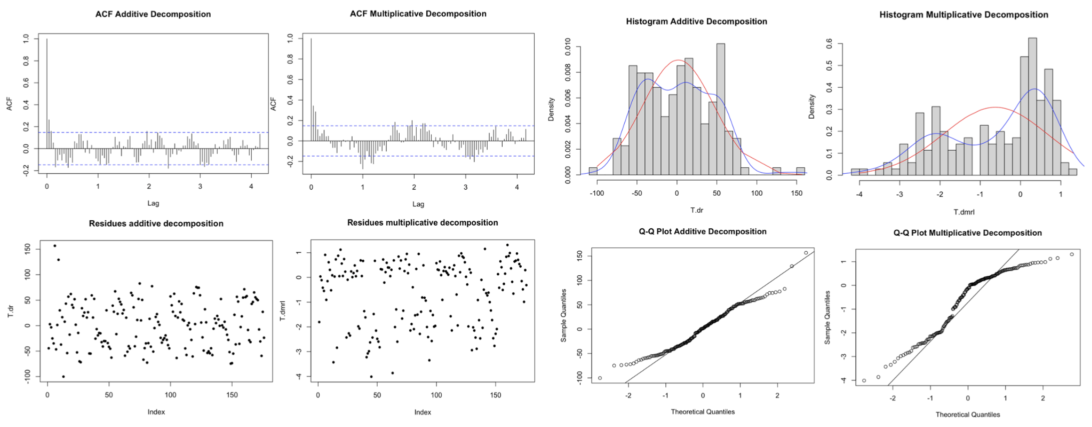


Figure 4.7: Decompositions' residues' comparison

Considering the scatter plot, the ACF, the histogram (by overlapping the empirical density and the theoretical gaussian density), the Q-Q plot, and the metrics highlighted in Table 4.3, the additive decomposition seemed to be the best decomposition method.

	Non-explained variance	Shapiro-Wilk test p-value	Autocorrelation function's variability
Additive decomposition	0.62	0.002	0.24
Multiplicative decomposition	0.64	1e-8	0.23

Table 4.3: Decompositions' residues' comparison

A.1.1 Forecasting methods

Different forecasting methods have been compared.
The *Holt-Winters method* with additive decomposition highlighted the following optimal values for the parameters:

Parameter	Value
α	0.32
β	0
γ	0.72
Initial intercept	-0.74
Initial slope	5.43

Table 4.4: Parameter choices for the Holt-Winters method

The initial intercept and slope were obtained with a linear regression.
A range of parameters close to the ones provided by the model was explored, but still, the previous values have been chosen to go on with the analysis.
The partial auto-correlation function of the time series was explored to build a *manual regression method*.

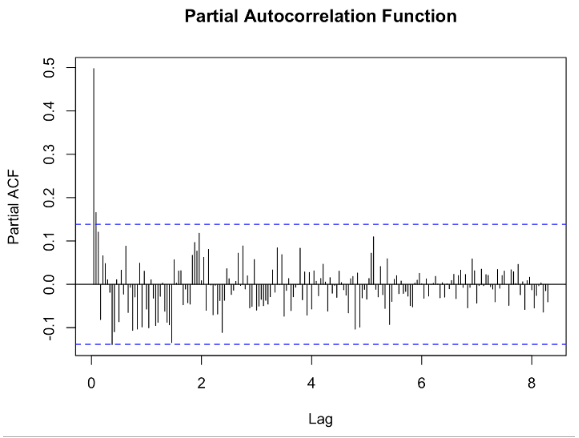


Figure 4.8: Partial ACF

The PACF highlighted two lags as possible interesting previous values to forecast the next one.
A *manual linear regression method* has been trained, though rejected due to the low value of the coefficient of determination.
Furthermore, the *Yule-walker regression method* has been considered to try to forecast our time series. Not realistic results were though obtained with this method.

The *Least squares method* determined that 23 lags could be the optimal number of lags to consider.
The Holt-Winters method and the Least squares method were compared using a residues analysis and a cross-validation, considering the mean square error as metric, and the Least squares method has been the best method to forecast the average total hourly CPU.

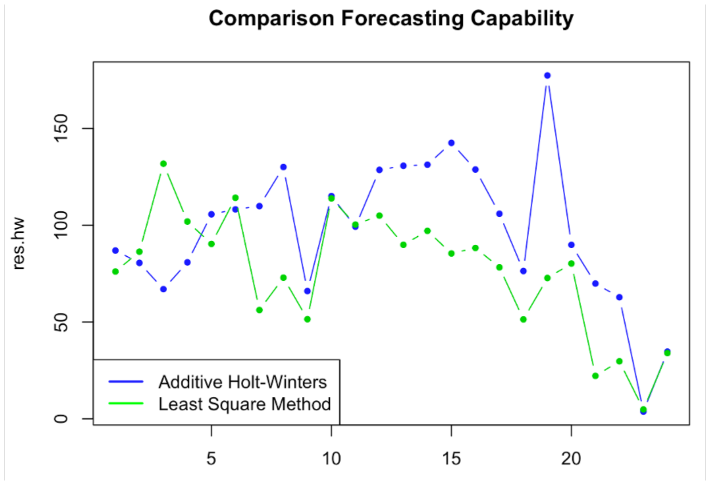


Figure 4.9: Cross-validation comparison

	Mean square error
HW method	47.40
LS method	36.96

Table 4.5: Mean square error CV comparison

A.2 Memory Time Series Analysis

The time series analysis of the average total hourly memory was performed using the same methodology as the one shown in the previous chapter, determining the Holt-Winters method with additive decomposition as the best forecasting method.

A.3 Failure classification

Different probabilistic classifiers have been compared for predicting the failure of the Glidein.
Performing a cross-validation comparison and using as metrics the accuracy and the ROC curve, the following results were obtained.

	Accuracy's Mean	Accuracy's Std	AUC
Linear Regression	0.33	0.07	-
Logistic Regression	0.84	0.04	0.90
Linear Discriminant Analysis	0.83	0.04	0.91
Quadratic Discriminant Analysis	0.85	0.04	0.90

Table 4.6: CV comparison

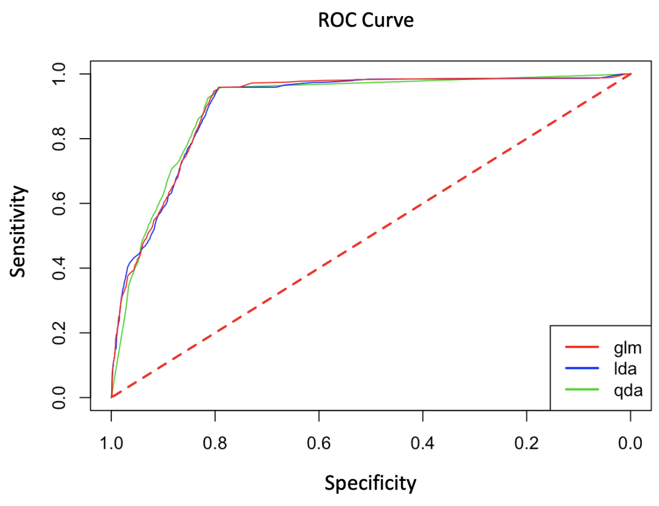


Figure 4.10: ROC Curves Comparison

After a site-by-site cross-validation and a robustness comparison, the logistic regression was chosen as the best classification method for our purpose, even if performance site-by-site were much more volatile than the general case, as indicated by the standard deviation.

	Accuracy's Mean	Accuracy's Std
Logistic Regression Method	0.75	0.23

Table 4.7: Site-by-site CV comparison

As previously stated, the probability threshold has then been varied in order to obtain a better trade-off between accuracy and sensitivity.