Table of Contents

Environment variables for multi-core jobs

This page describes the specification which has been deployed at a few sites. Please see the revised specification from January 2016 which the Task Force has produced and is preparing to create implementations of.

DRAFT 0.99 (15/05/2014)

Abstract

Within the HEPiX virtualization group a mechanism was discussed which allows to access detailed information about the current host and the current job by the job itself, as well as providing access to pilot job internals to the resource provide from a unified place on the worker node. This allows user payload to access meta information, independent of the current batch system, to access information like the performance of the node or calculate the remaining run time for the current job, and the resource provider to access internal pilot information, independent of the pilot framework, like the number of cores in use and estimated termination times.

Introduction

The proposed schema is made to be extensible so that it can be used to add additional information. The purpose of this document is to define the specifications and use case of this schema. It should be seen as the source of information for the actual implementation of the required scripts by the sites.

Definitions

When jobs are running within virtual machines, the entity that performs system level configuration within the VM (typically with root access) acts as the resource provider referred to in the rest of this document.

Environment variables

For each job, three environment variables may be set, with the following names:

Variable	Contents	Comments
MACHINEFEATURES	Path to a directory	Execution specific information
JOBFEATURES	Path to a directory	Job specific information
JOBSTATUS	Path to a directory	Internal job (pilot) information

These environment variables are the base interface for the user payload. They must be set for the job environment by the resource provider. In the case of virtual machines on IaaS cloud platforms, the resource provider may discover the values to set for the first two environment variables from the machinefeatures and jobfeatures metadata keys provided by the cloud infrastructure. These metadata keys should only be accessed once in the lifetime of each virtual machine.

Directories

The environment variables point to directories created by the resource provider. Inside, the file name is the key, the contents are the values, so that files can be referred to with expressions like $MACHINEFEATURES/shutdowntime . The directory name should not include the trailing slash. These directories are either local directories in the filesystem or sections of the URL space on an HTTP(S) server. The user positively determines whether the files are to be opened locally or over HTTP(S) by checking for a leading slash or the prefix http:// or https:// respectively. Typically this can achieved using library

functions which can transparently handle local files and remote URLs when opening files. The files may be accessed multiple times to check for changes in value or in the absence of caching by the user. The HTTP(S) server may provide HTTP cache control and expiration information which the user may use to reduce the number of queries.

The $MACHINEFEATURES and $JOBFEATURES directories contain files created by the resource provider.

The $JOBSTATUS directory is initially empty, and will be populated by the pilot job itself.

All files in the directories must be readable by both the user and the resource provider services.

Use cases

Use cases to be covered are

Identifier	Actors	Pre-conditions	Scenario	Outcome	(Optional) What to avoid
1.	user	job starter script	The job needs to calculate the remaining time it is allowed to run		
2.	user	job starter scripts	The job needs to know how long it was already running		
3.	user	host setup	The job wants to know the performance of the host in order to calculate the remaining time it will need to complete (for CPU intensive jobs)		
4.	site	host setup	A host needs to be drained. The payload needs to be informed of the planned shutdown time		
5.	site	job starter script	A multi-core user job on a non-exclusive node needs to know how many threads it is allowed to start. This is specifically of interest in a late-binding scenario where the pilot reserved the cores and the user payload needs to know about this.		
6.	site	job starter script	A user job wants to know how many job slots are allocated to the current job		
7.	site	job starter script	A user jobs wants to know the maximum amount of disk space it is allowed to use		
8.	site	job starter script	A user job wants to setup memory limits to protect itself from being killed by the batch system automatically		
9.	site	site batch system	The site wants to know how much longer the job will be running		
10.	site	site batch system	The site wants to know the amount of draining waste, if the job was asked to drain		
11.	site	site batch system	The site wants to know the amount of waste, if the job was killed		
12.	site	site batch system	The site wants to pick the job that is the least critical for the user		

Requirements

- The propose schema must be unique and leave no room for interpretation of the values provided.
- For this reason, basic information is used which is well defined across sites.
- Host and Job information can be both static (like the HS06 rating) and dynamic (eg shutdown time may be set at any time by the site).
- Pilot internal status information is all dynamic.
- Job specific files will be owned by the user and possibly reside on a /tmp like area

General explications

The implementation, that is the creation of the files and their contents can be highly site specific. A sample implementation can be done per batch system in use, but it is understood that sites are allowed to change the implementation, provided that the created numbers match the definitions given in this document.

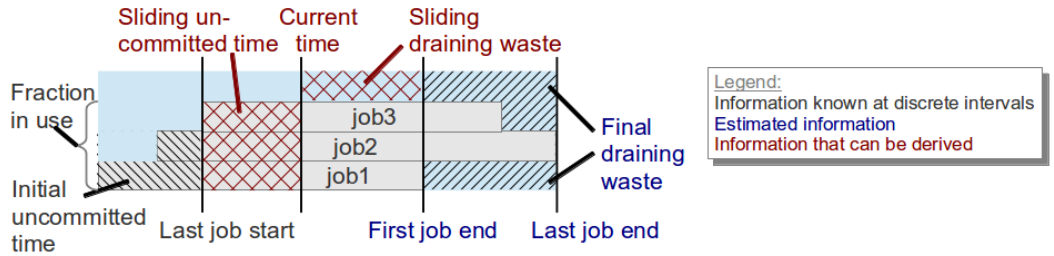
Clarification on terms

Normalization and CPU factors

At many sites batch resources consist of a mixture of different hardware types which have different performance.When a user submits a job to a queue, this queue typically sets limits on the CPU time and the wall clock time of the job. If such a job ends up on a faster node, it will terminate quicker. To avoid that jobs which run on slower nodes are terminated prematurely, CPU and Wall clock times are usually scaled with a factor which depends on the performance of the machine. This factor is called the CPU factor. A reference machine has a CPU factor of 1. For such a machine, normalized and real time values for CPU are the same. A CPU factor below 1 means that the worker node is slower than a given reference. In this case normalized times are larger than the real time values, and jobs are allowed to run longer in order to terminate. A CPU factor above 1 means that the worker node is faster than a given reference. In this case normalized times are smaller than the real time values.

Pilot status information

The proposed attributes cover the information that multi-job pilots own as part of their job scheduling activity. Below you can see a schematic view of this information; for more details, refer to the CHEP paper (published paper , talk)



All the information depicted above can be re-calculated at discreet intervals, e.g. state changes in the pilot:

- Number of CPUs in use at this time (labeled "Fraction in use" above, assumes each user job uses a fixed amount for its lifetime)
- Last time a user job started in the pilot (labeled "Last job start" above)
- The integral of useful wallclock that would be wasted, if the pilot would have been killed just when the latest job were about to be started (labeled "Initial uncommitted time")
- The estimated time the last job currently running will end. Note: this does **not** need to be the last job started.
- The earliest estimated time a job will end (labeled "First job end" above)

- The estimated integral of resources that would be wasted, if draining started immediately after the fits job ends (labeled "Final draining waste" above)

Most pilot jobs will typically start more jobs, if possible, thus extending any job termination estimates. There are however times when a pilot is ready to be retired, and does not expect more user jobs to be fetched. We provide an attribute to communicate this to the resource owner.

In addition, we want to allow a user to express a preference for one job versus any other job, in the form of a priority number.

List of requirements

Job specific information which are:

- found in the directory pointed to by $JOBFEATURES
- owned by the user who is executing the original job. In the case of pilots this would be the pilot user at the site.
- created before the user job starts, eg during a job starter script.

Identifier	File Name (key)	Originating use cases	Value	(Optional) Comments
1.1	cpufactor_lrms	1,3	Normalization factor as used by the batch system.	Can be site specific
1.2.1	cpu_limit_secs_lrms	1	CPU limit in seconds, normalized	Divide by **cpufactor_lrms** to retrieve the real time seconds. For multi-core jobs it's the total.
1.2.2	cpu_limit_secs	1	CPU limit in seconds, real time (not normalized)	For multi-core jobs it's the total.
1.3.1	wall_limit_secs_lrms	1	Run time limit in seconds, normalized	Divide by **cpufactor_lrms** to retrieve the real time seconds
1.3.2	wall_limit_secs	1	Run time limit in seconds, real time (not normalized)	
1.4	disk_limit_GB	7	Scratch space limit in GB (if any)	If no quotas are used on a shared system, this corresponds to the full scratch space available to all jobs which run on the host. Counting is 1GB = 1000MB = 1000^2kB
1.5	jobstart_secs	2	Unix time stamp (in seconds) of the time when the job started in the batch farm.	This is what the batch system sees, not when the user payload started to work.
1.6	mem_limit_MB	8	Memory limit (if any) in MB.	Total memory. Count with 1000 not 1024, that is 4GB corresponds to 4000
1.7	allocated_CPU	5	number of allocated cores to the current job	Allocated cores can be physical or logical
1.8	shutdowntime_job	1	dynamic value, shutdown time as a UNIX time stamp (in seconds)	optional, if the file is missing no job shutdown is foreseen. The job needs to have finished all its processing when the shutdowntime has arrived

Host specific information which are:

- found in the directory pointed to by $MACHINEFEATURES
- readable by the user who is executing the original job. In the case of pilots this would be the pilot user at the site.
- created before the user job starts

Identifier	File Name (key)	Originating use cases	Value	(Optional) Comments
2.1	hs06	3	HS06 rating of the full machine in it's current setup	Static value. HS06 is measured following the HEPiX recommendations. If Hyperthreading is enabled, the additional cores are treats as if they were full cores
2.2	shutdowntime	4	dynamic value, shutdowntime as a UNIX time stamp (in seconds)	Dynamic. If the file is missing, no shutdown is foreseen. The value is in real time, and must be in the future. Must be removed if the shutdowntime has arrived
2.3	jobslots	7	Number of job slots for the host	dynamic value, can change with batch reconfigurations
2.4	phys_cores	3	number of physical cores	-
2.5	log_cores	3	number of logical cores	can be zero if hyperthreading is off
2.6	shutdown_command	4	path to a command on the machine	optional, only relevant for virtual machines. A command provided by the site which provides a hook for the user to properly destroy the virtual machine and unregister it

Pilot status specific information which are:

- found in the directory pointed to by $JOBSTATUS
- owned by the user who is executing the original job. In the case of pilots this would be the pilot user at the site.
- created by the job, and will be updated several times during its lifetime

Identifier	File Name (key)	Originating use cases	Value	(Optional) Comments
3.1	used_CPU	10,11,12	Number of used cores by the job.	Must be locked before any of the other files in this section are either read or written to. Must be less or equal than allocated_CPU.
3.2	last_job_start	11	UNIX time (integer)	
3.3	first_exp_job_end	10	UNIX time (integer)	Good faith estimate
3.4	last_exp_job_end	9,10,12	UNIX time (integer)	Good faith estimate
3.5	last_max_job_end	9,10,12	UNIX time (integer)	Enforced limit
3.6	add_uncom_time	9,11	CPU seconds (integer)	

3.7	add_final_exp_waste	10	CPU seconds (integer)	Good faith estimate
3.8	can_postpone_last_job	9,10	string, either "True" or "False"	If the job decides to revert from "False" to "True", it should not update any of the other values for a significant amount of time.
3.9	priority_factor	12	Integer, higher is better	The semantics is user specific, and should not be used to compare jobs of different users.

See the introduction section for the explanation of semantics (while the names are shortened, they map 1-to-1 with the concepts).
Moreover, most of the above values are meant to be used together, so both readers and writers are requested to lock the **used_CPU** file before either reading or writing any of the files.

Notes:

Conversion formulas

Pilot status specific conversion formulas:

While the above information may be useful on their own, the pilot status one are usually combined to produce numbers used to make decisions.
Below are the formulas used to satisfy the originating use cases.

Originating use case	Symbolic name	Formula	(Op Con
9	remaining_time	$last_exp_job_end-$now or $last_max_job_end-$now	Ther two poss form depe on th of conf the u in th estin
10	draining_waste	($allocated_CPU-$used_CPU)*($first_exp_job_end-$now)+$add_final_exp_waste	
11	kill_waste	add_uncom_time+used_CPU*($now-$last_job_start)	

External Resources

A prototype for LSF exists and is installed at CERN. It needs to be adapted to follow the definitions in this document.
A prototype for HTCondor exists and is installed at UCSD.

Impact

User jobs and pilot frame works will have to be updated in order to profit from the above declarations.

Proposed extensions

Recommendations

Conclusions

The new mechanism allows to propagate basic information to and from user payload. The interface is independent of the batch system in use. The given information is designed to be sufficient to cover all mentioned use cases above.

References

- CERN prototype implementation for LSF (being implemented). This prototype will be CERN specific. It is likely that it will have to be adapted for use at other sites.
- Igor's presentation at CHEP 2013

-- UlrichSchwickerath - 10-Jul-2012

This topic: LCG > WMTEGEnvironmentVariables
Topic revision: r14 - 2016-01-21 - AndrewMcNab