The Standard Workload Format

The standard workload format (swf) was defined in order to ease the use of workload logs and models. With it, programs that analyze workloads or simulate system scheduling need only be able to parse a single format, and can be applied to multiple workloads.

The standard workload format abides by the following principles:

- The files are portable and easy to parse:
  - Each workload is stored in a single ASCII file.
  - Each job (or roll) is represented by a single line in the file.
  - Lines contain a predefined number of fields, which are mostly integers, separated by whitespace. Fields that are irrelevant for a specific log or model appear with a value of -1.
  - Comments are allowed, as identified by lines that start with a `;'. In particular, files are expected to start with a set of header comments that define the environment or model.
- The same format is used for models and logs. This implies that in each context certain fields may be redundant; for example, logs do not contain data about feedback and depdencies among jobs, which might appear in models, whereas models do not contain data about wait times in queues.
- The format is completely defined, with no scope for user extensability. Thus you are guaranteed to be able to parse any file that adheres to the standard, and multiple competing and incompatible extensions are avoided. If experience shows that important attributes have been left out, they will be included in the future by creating an updated version of the standard.

Versions

The first version to be published was version 2.

The current version, described here, is version 2.2.

Version	Additions
2.1	Add status 5 (canceled job)
	MaxProcs header comment to support SMP nodes
	Format for queue information to help automatic parsing
2.2	Add MaxQueues header comment
	Add MaxPartitions header comment
	Added TimeZoneString header comment
	Deprecated the buggy TimeZone header comment (replaced by TimeZoneString)
	Changed format of StartTime and EndTime be like that of 'date'

The Data Fields

1. **Job Number** -- a counter field, starting from 1.
2. **Submit Time** -- in seconds. The earliest time the log refers to is zero, and is usually the submittal time of the first job. The lines in the log are sorted by ascending submittal times. It makes sense for jobs to also be numbered in this order.
3. **Wait Time** -- in seconds. The difference between the job's submit time and the time at which it actually began to run. Naturally, this is only relevant to real logs, not to models.
4. **Run Time** -- in seconds. The wall clock time the job was running (end time minus start time).
We decided to use ``wait time" and ``run time" instead of the equivalent ``start time" and ``end time"

because they are directly attributable to the scheduler and application, and are more suitable for models where only the run time is relevant.
Note that when values are rounded to an integral number of seconds (as often happens in logs) a run time of 0 is possible and means the job ran for less than 0.5 seconds. On the other hand it is permissable to use floating point values for time fields.

5. **Number of Allocated Processors** -- an integer. In most cases this is also the number of processors the job uses; if the job does not use all of them, we typically don't know about it.
6. **Average CPU Time Used** -- both user and system, in seconds. This is the average over all processors of the CPU time used, and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average.
7. **Used Memory** -- in kilobytes. This is again the average per processor.
8. **Requested Number of Processors**.
9. **Requested Time**. This can be either runtime (measured in wallclock seconds), or average CPU time per processor (also in seconds) -- the exact meaning is determined by a header comment. In many logs this field is used for the user runtime estimate (or upper bound) used in backfilling. If a log contains a request for total CPU time, it is divided by the number of requested processors.
10. **Requested Memory** (again kilobytes per processor).
11. **Status** 1 if the job was completed, 0 if it failed, and 5 if cancelled. If information about chekcpointing or swapping is included, other values are also possible. **See usage note below.** This field is meaningless for models, so would be -1.
12. **User ID** -- a natural number, between one and the number of different users.
13. **Group ID** -- a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.
14. **Executable (Application) Number** -- a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.
15. **Queue Number** -- a natural number, between one and the number of different queues in the system. The nature of the system's queues should be explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.
16. **Partition Number** -- a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.
17. **Preceding Job Number** -- this is the number of a previous job in the workload, such that the current job can only start after the termination of this preceding job. Together with the next field, this allows the workload to include feedback as described below.
18. **Think Time from Preceding Job** -- this is the number of seconds that should elapse between the termination of the preceding job and the submittal of this one.

Consistency and data quality

It is recommended to take measures to ensure that the data is consistent, and this is indeed done for logs on this site. For example, it is verified that fields like wait time, runtime, and number of processors do not contain negative values, and that the number of processors specified does not exceed the number available in the system. However, such checks are not made for fields representing *requested>/i> resources, so for example the field of requested processors may indeed specify more than the number of processors available in the system*.

Data quality problems are discussed in the paper Experience with the Parallel Workloads Archive (Technical Report 2012-6).

Usage of the Status field

The main usage of the status field is to note the job's status. This isn't as straightforward as it sounds.

The simple case is jobs that complete normally, and have status 1.

The harder case is jobs that don't complete normally. This can happen for several reasons:

1. *The job failed (e.g. segmentation fault). This is given status 0.*
2. *The job was cancelled by the user (like ^C in Unix). This is given status 5. Note that cancelled jobs may have positive runtimes and processors if cancelled after they started to run, or 0 or -1 if cancelled while waiting in the queue.*
3. *The job was killed by the system (e.g. because it exceeded its requested run time). This may be given different status values in different logs; it will typically be 0 or 5, but might also be 1.*

Note also that the distinction between failure / cancellation / killing is not necessarily accurate, as the distinction typically does not appear in the original logs.

If a log contains information about checkpoints and swapping out of jobs, a job can have multiple lines in the log. In fact, we propose that the job information appear twice. First, there will be one line that summarizes the whole job: its submit time is the submit time of the job, its runtime is the sum of all partial runtimes, and its code is 0 or 1 according to the completion status of the whole job. In addition, there will be separate lines for each instance of partial execution between being swapped out. All these lines have the same job ID and appear consecutively in the log. Only the first has a submit time; the rest only have a wait time since the previous burst. The completed code for all these lines is 2, meaning ``to be continued''; the completion code for the last such line is 3 or 4, corresponding to completion or being killed. It should be noted that such details are only useful for studying the behavior of the logged system, and are not a feature of the workload. Such studies should ignore lines with completion codes of 0 and 1, and only use lines with 2, 3, and 4. For workload studies, only the single-line summary of the job should be used, as identified by a code of 0 or 1.

To summarize, the status field codes are (or should be) as follows:

0	Job Failed
1	Job completed successfully
2	This partial execution will be continued
3	This is the last partial execution, job completed
4	This is the last partial execution, job failed
5	Job was cancelled (either before starting or during run)

Usage of the preceding / think fields

The last two fields work as follows. Suppose we know that a.out, job number 123, should start ten seconds after the termination of gcc x.c, which is job number 120. We could give job number 123 a submittal time that is 10 seconds after the submittal time plus run time of job 120, but this wouldn't be right -- changing the scheduler might change the wait time of job 120 and spoil the connection. The solution is to use "preceding job" and "think time" fields to save such relationships between jobs explicitly. In this example, for job number 123 we'll put 120 in its preceding job number field, and 10 in its think time from preceding job field. In a workload log, it is possible to include both the submittal time and the precedence information; models can include precedences with dependent jobs that don't have independent arrival times.

Header Comments

The first lines of the log may (or rather, shold) be of the comments with the format ";Label: Value". These are special header comments with a fixed format, used to define global aspects of the workload. Predefined labels are:

1. ***Version**: Version number of the standard format the file uses. The format described here is version 2.*

2. ***Computer***: *Brand and model of computer*
3. ***Installation***: *Location of installation and machine name*
4. ***Acknowledge***: *Name of person(s) to acknowledge for creating/collecting the workload.*
5. ***Information***: *Web site or email that contain more information about the workload or installation.*
6. ***Conversion***: *Name and email of whoever converted the log to the standard format.*
7. ***MaxJobs***: *Integer, total number of jobs in this workload file.*
8. ***MaxRecords***: *Integer, total number of records in this workload file. If no checkpointing/swapping information is included, there is one record per job, and this is equal to MaxJobs. But with chekpointing/swapping there may be multiple records per job.*
9. ***Preemption***: *Enumerated, with four possible values. 'No' means that jobs run to completion, and are represented by a single line in the file. 'Yes' means that the execution of a job may be split into several parts, and each is represented by a separate line. 'Double' means that jobs may be split, and their information appears twice in the file: once as a one-line summary, and again as a sequence of lines representing the parts, as suggested above. 'TS' means time slicing is used, but no details are available.*
10. ***UnixStartTime***: *When the log starts, in Unix time (seconds since the epoch)*
11. ***TimeZone: DEPRECATED and replaced by TimeZoneString***.
    A value to add to times given as seconds since the epoch. The sum can then be fed into gmtime (Greenwich time function) to get the correct date and hour of the day. The default is 0, and then gmtime can be used directly. Note: do not use localtime, as then the results will depend on the difference between your time zone and the installation time zone.
12. ***TimeZoneString***: *Replaces the buggy and now deprecated TimeZone.*
    TimeZoneString is a standard UNIX string indicating the time zone in which the log was generated; this is actually the name of a zoneinfo file, e.g. ``Europe/Paris''. All times within the SWF file are in this time zone. For more details see the usage note below.
13. ***StartTime***: *When the log starts, in human readable form, in this standard format: Tue Feb 21 18:44:15 IST 2006 (as printed by the UNIX 'date' utility).*
14. ***EndTime***: *When the log ends (the last termination), formatted like StartTime.*
15. ***MaxNodes***: *Integer, number of nodes in the computer. List the number of nodes in different partitions in parentheses if applicable.*
16. ***MaxProcs***: *Integer, number of processors in the computer. This is different from MaxNodes if each node is an SMP. List the number of processors in different partitions in parentheses if applicable.*
17. ***MaxRuntime***: *Integer, in seconds. This is the maximum that the system allowed, and may be larger than any specific job's runtime in the workload.*
18. ***MaxMemory***: *Integer, in kilobytes. Again, this is the maximum the system allowed.*
19. ***AllowOveruse***: *Boolean. 'Yes' if a job may use more than it requested for any resource, 'No' if it can't.*
20. ***MaxQueues***: *Integer, number of queues used.*
21. ***Queues***: *A verbal description of the system's queues. Should explain the queue number field (if it has known values). As a minimum it should be explained how to tell between a batch and interactive job.*
22. ***Queue***: *A description of a single queue in the following format: queue-number queue-name (optional-details). This should be repeated for all the queues.*
23. ***MaxPartitions***: *Integer, number of partitions used.*
24. ***Partitions***: *A verbal description of the system's partitions, to explain the partition number field. For example, partitions can be distinct parallel machines in a cluster, or sets of nodes with different attributes (memory configuration, number of CPUs, special attached devices), especially if this is known to the scheduler.*
25. ***Partition***: *Description of a single partition.*
26. ***Note***: *There may be several notes, describing special features of the log. For example, ``The runtime is until the last node was freed; jobs may have freed some of their nodes earlier''.*

Usage of the TimeZone and TimeZoneString fields

The TimeZone header comment is DEPRECATED and replaced by TimeZoneString.

- *TimeZoneString is a standard UNIX string indicating the time zone in which the log was generated.*

- *All specified epoch times within the SWF file relate to this time zone, e.g. in the LPC log, UnixStartTime=1091615532 (= seconds since epoch) translates to "Wed Aug 04 12:32:12 CEST 2004" in France time (where the log was generated).*
- *Indeed, the TimeZoneString of the LPC log is Europe/Paris. This must be the name of a standard zoneinfo file, which is usually found under the /usr/share/zoneinfo/ directory in UNIX systems (that is, the file /usr/share/zoneinfo/Europe/Paris must exist).*
- *If you want to convert an SWF time to local a time (e.g. to know the time-of-day in which a job J was submitted), then in Perl (for example) you do the following:*

```
use POSIX;
$ENV{TZ} = 'US/Pacific';  # a file under /usr/share/zoneinfo/
POSIX::tzset();           # new timezone takes effect
my $UnixStartTime = ...;  # start time of log in seconds since epoch
my $submit        = ...;  # SWF submit time of J
my ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst)
    = localtime($UnixStartTime + $submit);
```

- *If you want to do the same in C:*

```
#include <time.h>
#include <stdlib.h>

setenv("TZ", ":Europe/Paris", 1/*overwrite*/);
tzset();

time_t unixstart = ...; // UnixStartTime
time_t submit    = ...; // SWF submit time of J
time_t sum       = unixstart + submit;

struct tm *datetime;
datetime = localtime( &sum );
```

- *The reason TimeZone is buggy and was replaced by TimeZoneString is that with the former we completely ignore daylight saving periods.*
- *Note that in contrast to TimeZone, if we use TimeZoneString then we should use localtime (not gmtime).*
- *Also, note that contrary to common belief, the epoch is defined to be 00:00:00 on January 1, 1970, in London (that is, it's not true that every time-zone counts seconds since 00:00:00 on January 1, 1970 on that time zone).*

Parsing

To support usage of the standard workload format, we have an example Perl script for parsing it.

Acknowledgements

The standard workload format was proposed by David Talby and refined through discussions with Dror Feitelson, James Patton Jones, and others.

If you use it, you can refer to the following paper:
Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, ``Benchmarks and Standards for the Evaluation of Parallel Job Schedulers''. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, Lect. Notes Comput. Sci. vol. 1659, pp. 66-89.
Don't forget to also point to this web page, which contains the most up-to-date version of the definition.

The improvements in handling time zones are due to Dan Tsafrir.

Back to the Parallel Workloads Archive home page

feit@cs.huji.ac.il / Feb 21, 2006