A Standard Timestamp for Grid Computing (DRAFT)

Dan Gunter, Brian Tierney
LBNL

1.0 Motivation

There are many occasions for producing and consuming timestamps in computing, ranging from performance analysis to security protocols. Grid applications and protocols which produce and consume timestamps need to interoperate, either in real-time or by using and storing information in directory services or archives. A standard for timestamp representation needs to be defined which satisfies these requirements. The many different ad-hoc timestamps now in use should be replaced with a single “standard” format or formats, thus allowing the solutions to the problems of representing dates, times, precision, and accuracy to be reused throughout the Grid developer community.

2.0 Scope

This document proposes a *model* and *format* for timestamps. It does not make recommendations on how to gather and verify the accuracy of underlying timestamp values. At the time of this writing, a *best practices* document for creating timestamps does not exist, although this would be a valuable addition to the Grid Forum deliverables.

This document also does not attempt to model any other metric of time aside from a single moment. More complex constructions involving time, such as time intervals or time series, should be discussed in a separate document.

Finally, the timestamps presented here are primarily intended to represent real measured times, such as those returned from a host’s clock, or a time logged into a system log or web log. Applications which use time as part of a mathematical computation or scientific simulation will continue to use their proper formats.

3.0 Timestamp Model

A timestamp is an estimate of a single moment in time. In most applications, the timestamp describes when a real-world event occurred, and thus the timestamp is only an estimate of the “real” time of the event. In order for this estimate to be used in analyses, it should carry with it an estimate of its own *precision* and *accuracy.* Thus, the timestamp has three parts: time value, precision, and accuracy. Each of these will be discussed in more detail below.

3.0.1 Time Zones

The time of day which most people are familiar with is really composed of two things: a time, and a time zone. The time zone is the offset of the time from Universal Coordinated Time (UTC), which is the more precise standard replacing the perhaps more familiar Greenwich Mean Time (GMT).

In order to avoid the overhead of adding and subtracting time zones from timestamps in the intermediate processing stages, timestamps for the Grid should always be in UTC time.

3.0.2 Timestamp Value

The timestamp value is measured in seconds and fractions of a second UTC. At least nanosecond precision should be possible, and dates should be representable at least within the range of the current UNIX timestamp (1970-2032).

3.0.3 Timestamp Precision

The *precision* of a timestamp indicates how many "seconds per tick" were performed by the underlying clock. The representable range must be at least 10^{-10} seconds (picoseconds) to 10^{2} seconds (~1.5 minutes). The upper end of this range is somewhat arbitrary, but it is suspected that almost all precisions will be sub-second. The precision is often determined by the underlying operation system and the programming language being used. For example, the standard timing call in C return a precision of microseconds, where Java, Perl, and Python all return a value with a precision of milliseconds. Some operating systems also provide access to a nanosecond clock. (e.g.: the Solaris and RTLinux gethrtime() call).

The precision of the timestamp is represented as an 8-bit signed integer representing the logarithm base two (2) of the number of seconds per tick. Thus, 2^{-128} to 2^{127} seconds per tick can be represented. In practical terms, this means that a positive number is the number of low-order bits which are insignificant in the timestamp's seconds field, and a negative number is the number of significant bits in the fraction of seconds field.

3.0.4 Timestamp Accuracy

The *accuracy* of a timestamp summarizes the how close the reported value is to the "true" value, or the "margin of error" for the time value. The accuracy of a timestamp is represented in seconds, with a range of at least 10^{-10} seconds to 10^{3} seconds.

Unfortunately it can be quite difficult to determine what the accuracy of a timestamp is. For hosts running NTP, the *xntpdc* program reports its notion of accuracy relative to a GPS time source, and is typically around 1 millisecond. For hosts running other forms of time synchronization, for example *rdate* or *timed*, the accuracy is probably be somewhere between 1 millisecond and .1 second. Additionally, these values only give the accuracy relative to the host they are synchronizing with, which may or may not be accurate. For hosts running without any time synchronization service, the accuracy is *null*.

This is a critical issue for timestamps in the Grid. We recommend that all hosts run an NTP daemon [4], and that all monitoring services check that this daemon is running and check what accuracy is reported by this daemon. Any Grid timestamp that does not include accuracy information should be assumed to have an accuracy of *null*. The truly difficult (and perhaps unsolvable) issue is how to ensure that programs that are generating timestamped events fill in their accuracy field accurately.

Determining the true accuracy of the timestamp, as well as any other measurement (e.g.: CPU load) is not trivial [5]. For this reason we suggest that the accuracy value here be from a standard "accuracy table" that the members of the Grid Performance working group build and evolve over time. This table would represent the "typical" accuracy for a given type of measurement. Grid measurement values could also include a pointer to a more complex XML description of the accuracy, which could include information such as number of samples, maximum, minimum, and standard deviation of the error measurements, sets of confidence intervals, and so on. (NOTE: if anyone has any further ideas on a good way to handle this, please let us know!)

4.0 Representations

The representation of timestamps must take into consideration three factors: compactness of external representation, ease of conversion to and from internal representation, and ease of representation in enclosing protocols. The third factor refers to the existence of enclosing protocols which cannot easily carry non-textual data (e.g. LDIF, XML). Balancing this factor with the desire for compactness and ease of conversion has led to a split of the representation into two types: binary and ASCII.

4.1 Binary Representation

The binary representation is compact and efficient, and should be used wherever the enclosing protocol or human readability do not preclude it. In this section, each part of the representation will first be discussed separately, then all three parts will be combined.

4.1.1 Binary Timestamp Value

The binary timestamp value is represented as two 32-bit unsigned integers in network byte order, similar to but not identical to the Network Time Protocol format [3]. The first integer represents seconds since 1/1/1970 (as returned by the Unix *gettimeofday* system call), the second number represents fractions of a second. Unlike NTP, the binary timestamp format presented here has a separate precision field, so the fractions of a seconds does not need to have zeros (0) in non-significant digits. Only UTC time is allowed, i.e. no timezone information is carried with the value.

4.1.2 Binary Timestamp Precision

The precision of the timestamp is represented as an 8-bit signed integer representing the logarithm base two (2) of the number of seconds per tick. In practical terms, this means that a positive number is the number of low-order bits which are insignificant in the timestamp's seconds field, and a negative number is the number of significant bits in the fraction of seconds field.

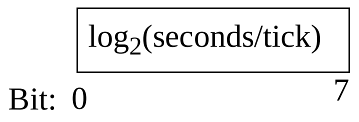

FIGURE 1. Binary timestamp precision representation

4.1.3 Binary Timestamp Accuracy

The accuracy of the timestamp is the number of multiples of the precision on either side of the timestamp value which bound its "likely" value (see "Timestamp Accuracy"). This is represented as an unsigned 32-bit integer. A *null* accuracy is represented as all 1's.

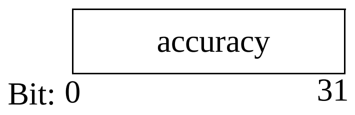

FIGURE 2. Binary timestamp accuracy representation

In order to calculate the accuracy as a number of seconds (or fractions of a second), the precision value should be used as a bit-shift to the accuracy value.

4.1.4 Binary Timestamp Format

The binary timestamp format combines the three representations above, in the same order, prefixed with an 8-bit header that has 4 bits for versioning and 4 bits for future use. The version of any timestamp conforming exactly to this document is zero (0), and the version field should be correspondingly all zeros. The 4 bits for future use should be left as zero (0).

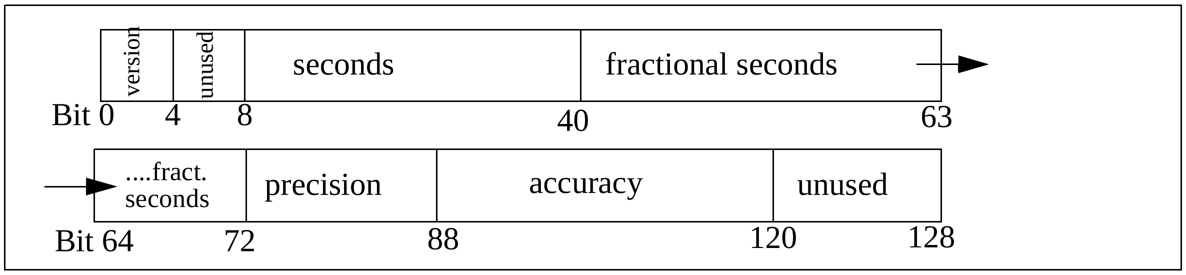

Figure 3: Overall binary timestamp format

4.1.5 Examples

- November 27, 2000 at 11:21am and 26.901 seconds (UTC), with a precision of roughly 1 millisecond (rounded to 2^{-10} seconds) and an accuracy of plus or minus one half of a second (500 milliseconds), would be represented as (in binary):

0	0	975320431	3869765534	-10	500

Figure 4: Sample Binary Message

- The same time as above with microsecond precision (again, rounded to 2^{-20} seconds) and an accuracy of plus or minus one-half of a second (500000 microseconds) would be represented as (in binary):

0	0	975320431	3869765534	-20	500000

Figure 5: Sample Binary Message

- The same time with precision of one second and no accuracy would be represented as (in binary):

0	0	975320431	3869765534	0	-4294967296

Figure 6: Sample Binary Message

4.2 ASCII Timestamp

Sometimes readability is more important in a timestamp than compactness of representation and ease of computer manipulation. This is obviously true when the ultimate consumer of the timestamp is a person; it is also sometimes true when the debugging process might involve direct observation of the timestamp data. Finally, certain commonly used application envelopes, for example LDIF and anything based on XML, are not optimized for binary data. In these cases, and especially when the volume and frequency of timestamps will not pose a significant load on the system, an ASCII timestamp may be used. In contrast to the binary timestamp, each byte in the ASCII timestamp will contain only printable ASCII characters.

In this section, each part of the representation will first be discussed separately, then all three parts will be combined in a final sub-section entitled “Timestamp Format”.

4.2.1 ASCII Timestamp Value

There are two good candidates for an ASCII timestamp representation. The first is a calendar-style representation, for example: 2000-10-26-08:34:26.30323. The second format is on which the year, month, day, etc. are a single concatencated string , with a decimal number of seconds, for example the ULM [1] format: 20001026083426 30.236553. Although the latter is slightly easier for a computer to generate and parse, the primary purpose of this timestamp format is readability, and the former representation is better for visual inspection.

Thus, we recommend a Grid timestamp value be represented using <u>one</u> of the possible variations on the ISO8601 time standard [2], shown below:

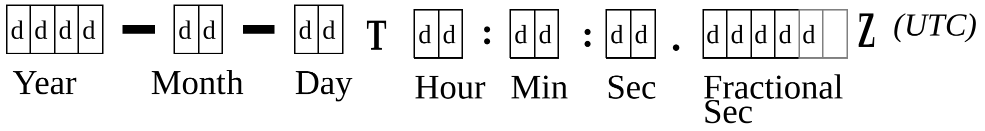

Figure 7: ASCII timestamp value representation

For example, 8:34am, 26 seconds, and 350 milliseconds on October 26th, 2000 UTC would be represented as: “2000-10-26T08:34:26.350Z”. The “Z” at the end signifies “UTC”, and is required. The length of the fractional seconds after the decimal point is from 1 to 10, with the common values expected to be 3, 6, and 9 digits.

4.2.2 ASCII Timestamp Precision

Precision is indicated in seconds. Typical values will be 1 (second), .001 (milliseconds), .000001 (microseconds) and .000000001 (nanoseconds). An optional letter “p” can be used to make the precision value more obvious to the eye.

For example, a timestamp with a precision of 10 seconds would read: “2000-10-26T08:34:26Zp10”.

If the precision is absent, but the timestamp contains a fractional second value, the number of digits in the fractional seconds is the implied subsecond precision. Thus, a date with millisecond precision could be represented in one of three ways: “2000-10-26T08:34:26.010Z”, “2000-10-26T08:34:26.01Zp.001”, or “2000-10-26T08:34:26.01Z.001”.

4.2.3 ASCII Timestamp Accuracy

The decimal number for the accuracy is placed after an “a”, and may have a decimal point and up to 10 digits on either side:

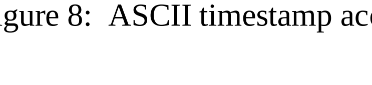

Figure 8: ASCII timestamp accuracy representation

4.2.4 ASCII Timestamp Format

The best way to summarize the ASCII timestamp format is with a few examples:

- October 26, 2000 at 8:34am and 26 seconds, with a precision of 1 millisecond and an accuracy of plus or minus one half of a second, would be represented as:

  2000-10-26T08:34:26Zp.001a.5

- January 1, 2001 at 3:12pm and 5 seconds, with a precision of 5 seconds and an accuracy of plus or minus 10 minutes, would be represented as:

  2001-01-01T15:12:05Zp5a600

- August 26, 1970 at 12:00 noon and 20 seconds and 356,675 microseconds, with a precision of 1 nanosecond and an accuracy of plus or minus 10 microseconds, would be represented as:

  1970-08-26T12:00:20.356675Zp.000000001a.00001

- October 26, 2000 at 8:34am and 26 seconds, with no precision or accuracy specified would be represented as:

  2000-10-26T08:34:26Z

5.0 Summary

This document has presented two ways of representing the same underlying information about a timestamp: a binary and an ASCII format. The binary format is clearly much more compact, and the ASCII format is clearly much more readable. Neither format is particularly complex. It is conceivable that a single small library could be written which would automatically format the system time in one or both of these formats. The same library could also convert a representation of a timestamp into an internal structure such as a Java *class* or a C *struct*. It is the intention of this document to provide a standard Grid timestamp format that is clear and unambiguous enough that independent conforming implementations of such a library would be able to interoperate.

6.0 References

[1] Abela, J., T. Debeaupuis, “Universal Format for Logger Messages”, IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-abela-ulm-05.txt

[2] ISO-8601, “Data Elements and Interchange Formats - Information Exchange - Representation of Dates and Times”, International Organization for Standardization, 1888 http://www.iso.ch/markete/8601.pdf

[3] Mills, D.L. “Internet time synchronization: the Network Time Protocol”. IEEE Trans. Communications COM-39, 10 (Octoberv1991), 1482-1493.

[4] Mills, D., Simple Network Time Protocol (SNTP). RFC 1769, March 1995.

[5] Wolski, R., Spring, N., Hayes, J., “Predicting the CPU Availability of Time-shared Unix Systems”, Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8), August, 1999.