Grid Timestamps: the Leap Second Problem

John Ryan, Trinity College Dublin, john.p.ryan@cs.tcd.ie
Brian Coghlan, Trinity College Dublin, coghlan@cs.tcd.ie

Abstract: It has been observed that there is potential for multi-second errors in the interpretation of Grid timestamps. Here we examine these issues and explore possible solutions.

Introduction

In prototyping the use of a proposed GGF timestamp format [Gunter and Tierney 2000], it was observed that there is potential for multi-second errors in the interpretation of Grid timestamps. This is due to the insertion of leapseconds into the UTC timescale in order to keep it in check with the UT1 and TAI systems, and the fact that some operating systems are unable to handle these events.

From the java.util.Date Class documentation [Sun 2001]:

> *"Although the Date class is intended to reflect Co-ordinated universal time (UTC), it may not do so exactly, depending on the host environment of the Java Virtual Machine. Nearly all modern operating systems assume that 1 day = 24 * 60 * 60 = 86400 seconds in all cases. In UTC, however, about once every year or two there is an extra second, called a "leap second." The leap second is always added as the last second of the day, and always on December 31 or June 30. For example, the last minute of the year 1995 was 61 seconds long, thanks to an added leap second. "*

Since the year 1995 lasted longer (due to aggregate deceleration of the earth due by tidal forces…), the UTC sequence of times should be:

31-December-1995 23:59:58
31-December-1995 23:59:59
31-December-1995 23:59:60 (+1s)
01-January -1996 00:00:00

The most widely used UNIX application to synchronize computers is NTP [NTP]. When the time comes for the insertion of a leapsecond the NTP kernal module introduces a leapsecond into the system time by repeating the second at which the leapsecond was introduced. The previous sequence would become:

31-December-1995 23:59:58	1000000000
31-December-1995 23:59:59	1000000001
31-December-1995 23:59:60 (+1s)	1000000001
01-January -1996 00:00:00	1000000002

As soon as the second is inserted, the fact that an insertion took place is forgotten about. If we were asked to calculate the duration from 31-December-1995 23:59:58 to 01-January-1996 00:00:00 given the corresponding system times, the answer would be 2 seconds, whereas it actually is 3 seconds.

However, some implementations of the java.util.Date class may include the leapseconds, some may not. Again, from the java.util.Calender Class Documentation:

> *"A second is represented by an integer from 0 to 61; the values 60 and 61 occur only for leap seconds and even then only in Java implementations that actually track leap seconds correctly."*

One can only speculate about the other useful class, java.sql.Date. From the java.sql.Date Class documentation:

> *"A thin wrapper around a millisecond value that allows JDBC to identify this as a SQL DATE."*

As the grid may comprise many different Java implementations it is clear that a solution was required that would work for all java implementations.

Existing Solutions

Similar issues exist with C libraries (e.g. *ctime*), but the GGF Performance Working Group [PerfWG 2001] has been sidestepping them so far. There is a C library available, libtai, provided by [Bernstein], but no Java implementation, and certainly no implementation where the grid as a whole can manage the insertion of leapseconds.

Proposed Solution

Below we outline a Java-oriented solution for managing the insertion of leapseconds on the grid. This solution is based on the use of the R-GMA grid information system [R-GMA] developed within the European DataGrid project [DataGrid], and indeed is now part of R-GMA.

Use of Time on the Grid

In a Grid context, in the generality, the information system should associate each measurement with the time it was made. It should handle fresh and archival data in a uniform manner, i.e. all information should be carry a time attribute. All sites must have the same view of time. All representations of time should include the times at which leapseconds were introduced, and all conversions between representations should honour the leapseconds.

At present the GGF promotes a revised time representation [Gunter and Tierney 2001] that attempts to enforce consistency by sticking to only one, ASCII, representation. The problem only occurs when there is a conversion between this representation and another, say binary, representation, or vice versa. Unless all the nodes involved in a grid computation use the same implementation, some will convert between time representations differently to others. This will yield multi-second errors in their interpretation of time.
As an example, let us assume node A is running the GridScheduler and doing information gathering from nodes B and C, and that node C correctly handles leapseconds, whereas node B doesn't (maybe it is a CRAY not a Linux cluster, with a different Java implementation). Let us assume B and C represent time internally as binary UNIX-time (milliseconds since 1st January, 1970) but interchange time in ASCII date format via XML. Node A will correctly receive node C's time, whereas it will receive an incorrect time from node B that is several seconds ahead of the actual time at node B. Note that this is independent of whether or not node A correctly handles leapseconds. The inverse scenario is presented in Figure 1.

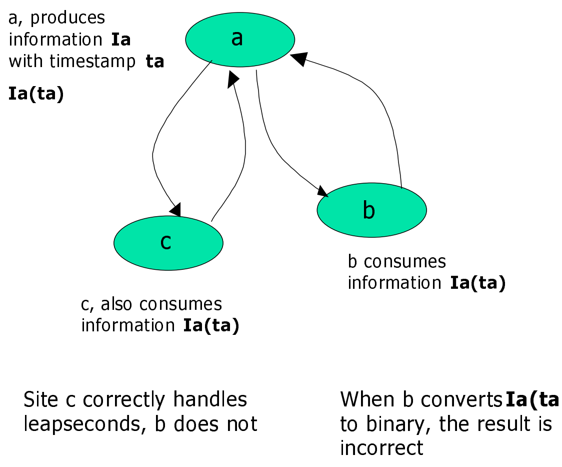
a, produces information **Ia** with timestamp **ta**
Ia(ta)
a
c
b
b consumes information **Ia(ta)**
c, also consumes information **Ia(ta)**
Site c correctly handles leapseconds, b does not
When b converts **Ia(ta)** to binary, the result is incorrect

Figure 1. Time conversion example (time exchanged via XML)

Clearly this will only affect fine-grain analysis, but this may be very important once network bandwidths increase, and also once nodes begin to exchange information and control messages more frequently.

How can this to be prevented?

There is no obvious way to overcome the problem short of an implementation that specifically ensures a uniform treatment of leap seconds across the grid.

A good first step would be a document providing the formulae for converting from UTC date to seconds since the epoch, and vice-versa, with leap seconds accounted for. These formulae could be tested against various libraries to determine which ones provide for leap seconds. Most probably do not. Surprisingly, for instance, POSIX mandates that the returned time *not* account for leap seconds (at least according to the comments in the NetBSD source code) [Gunter 2001].

Time

How is time defined? Current time standards are based on the rotation of the earth on its axis (e.g. *Universal Time*, UT/UT0/UT1/UT2), or on an atomic resonance (e.g. *Temps Atomique International*, TAI, aka Atomic Time). The S.I. second is defined as 9192631770 cycles of a caesium resonance. *Co-ordinated Universal Time* (UTC) is defined to exhibit the same duration of second as an S.I. second (i.e. as TAI), but with its count of seconds adjusted to include steps (leapseconds) that keep it within 0.9s of UT1.

UT1, UTC and TAI are clearly related [USNO]:

$$UTC = TAI - n*(leapseconds)$$
$$UTC - UT1 < 0.9s$$

UTC started on 1st January 1972 with n=10. UTC is irregularly kept within 0.9s of UT1 by adding these leapseconds. The IERS in Paris [USNO] determines when to introduce positive or negative leapseconds. This is a very irregular event, introduced as extra or absent seconds at end of a month (currently only at the end of June or December). A current list of leapsecond insertions is available from site [USNO ftp].

Epoch

How is time referenced? It is always done in relation to the *epoch* of the time standard. Each standard has a different epoch, as shown in Table 1.

Time Standard	Epoch
NTP	1st January, 1900
TAI	1st January, 1958
UNIX	1st January, 1970
UTC	1st January, 1972
GPS	1st January, 1980
Y2K	1st January, 2000

Table 1. Standard epochs

What are Timestamps?

Timestamps are attributes assigned to objects (such as measured values) to indicate time. The semantics of this depends on the application; it can indicate the time of measurement, the time of archival, the time that the timestamp was assigned, or whatever, so in all cases it only means what its author intended. There are two obvious timestamp representations: The first, an ASCII timestamp, is a formatted string, e.g. '2002-01-16T12: 45:50.36Z'. The second, a binary timestamp, is a local binary representation, perhaps in the form used by the resident operating system, e.g. for Linux, two integers or long integers.

ASCII Timestamp Formats

A number of efforts are currently underway to define a universally acceptable ASCI timestamp. The IETF currently has an RFC for defining date and time formats for the internet, based on the standard representation of dates and times in the ISO8601 standard. The GGF also has a standard model, more or less the same, but including meta-information such as precision and accuracy.

The IETF timestamp is defined as a formatted string: 'yyyy-mm-ddThh:mm:ss.ffffZ', where T and Z specify UTC, and are compulsory, yyyy-mm-dd describe the year, month and day, and hh:mm:ss.ffff describe the hour, minute, second and fractional second. The GGF format [Gunter and Tierney 2001, Gunter et al 2001] includes these plus optional accuracy and resolution fields, i.e. 'yyyy-mm-ddThh:mm:ss.ffffZjjjrkkka', where jjj is the resolution, and kkk the accuracy, e.g. '2002-01-16T12: 46:48.7567Z.00000001r.00001a'.

Binary Timestamp Formats

Traditionally UNIX *time* has been represented as a binary integer counting milliseconds since the 1st January 1970. A new UNIX structure *timeval* has now been widely implemented, using two long integers: the first is a 64-bit signed second value, and the second a 64-bit signed microsecond value, such that *timeval* is a count of microseconds since 1st January, 1970. It is almost certain that the microsecond resolution is not sufficiently precise for future applications. 'Buyer beware' also applies: resolution can be constrained by various platforms and programming languages, in Java for instance, *time* is counted in milliseconds.

Leapsecond Correction

As previously stated, problems arise when there is a conversion between one representation and another, such as between the IETF ASCII and the UNIX binary representations, or vice versa. Here we propose a grid solution where one known site, and any mirrors, accumulate leapsecond information from an official source, and publish this to the grid information system. All other sites then obtain the leapsecond information from the information system, and use this to correctly convert between timestamp formats.

ASCII to binary conversion

Let us assume we wish to convert an UTC ASCII timestamp t_a to a TAI binary timestamp t_b. First find the corresponding leapsecond value, L, for t_a. Then calculate the corrected binary value as $(t_b = 60*Secs + 3600*Mins + \ldots + L)$.

Binary to ASCII ->conversion

Assume we wish to convert a TAI binary timestamp t_b to an UTC ASCII timestamp t_a. First find the corresponding leapsecond value, L, for t_b. The corrected ASCII value t_a is derived from $(t_b - L)$.

Prototype Solution

A prototype has been constructed, see Figure 2, where the information system is the European DataGrid R-GMA [R-GMA], and the leapsecond data is acquired from the USNO ftp site [USNO ftp] then published by a *DBProducerServlet*. A UNIX *cron* job causes re-acquisition every month. All other sites obtain the leapsecond information from this producer via the information system. A static Java timestamp conversion class is then instantiated that uses the published information to correctly convert between formats.

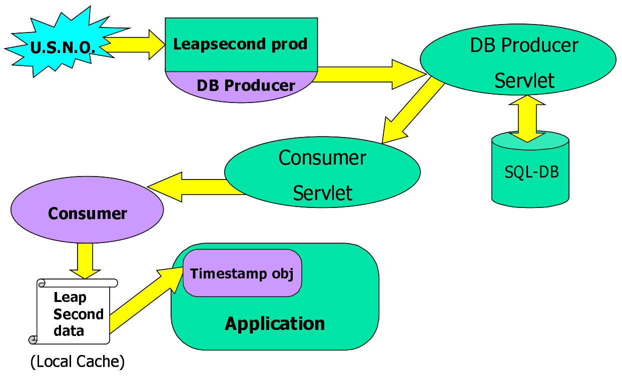

Figure 2. Prototype solution

A caching strategy is employed to hide the latency of the information system. The *timestamp* class prefetches leapsecond data from the information system into a local cache file; this happens when the object is instantiated. Another *cron* job refreshes the cache every day. Using a file ensures the cache automatically adapts to the number of leapseconds since 1972 (22 so far).

Performance is quite good even without optimisation. On a 450MHz Pentium II with the Sun JDK, there is a ~20mS start-up delay when a *timestamp* object is instantiated, i.e. while reading from the cache file. Thereafter it takes 225 sec to perform 10^6 ‘double’ timestamp conversions, from ASCII to binary and back to ASCII, implying an average conversion time of 112 microseconds. The prototype has been verified for multiple positive and negative leapseconds. For a live demonstration, see http://www.cs.tcd.ie/coghlan/meta/datagrid/timestamps/

At present the *cron* jobs occur at fixed times. The first planned optimisation is that these times will be randomised about their fixed values.

Grid Timestamps

The above solution is viable, but there are two outstanding decisions to be made:

(a) The fractional-second formats, both binary and ASCII, must be chosen. The IETF only have an ASCII decimal fraction of variable length. The GGF ASCII format is based on the IETF format. The GGF originally proposed a 32bit binary fraction within a binary timestamp format, but subsequently regressed to the ASCII format only. The

UNIX binary fraction is given as the elapsed microseconds, which is inadequate for future applications.

(b) The epoch, both binary and ASCII, must be chosen. The UNIX epoch (1st January 1970) is unsatisfactory, since it leads to non-integer leapsecond values. The most sensible epoch is either TAI since it is the base atomic standard, UTC since it is the base leapsecond-corrected standard, or Y2K since it is closest to the start of the grid era (but the earlier epochs would at least encompass experiments over the intervening decades).

The prototype supports the GGF ASCII timestamp format (enhanced UTC). As TAI represents time as a simple count of seconds since the epoch (UTC does not), it is sensible for binary time representations. Hence the prototype supports binary timestamps that employ a 64bit-integer count of seconds since the TAI epoch, a 64bit-integer count of picoseconds, and 64bit integer picosecond resolution and accuracy values. At the risk of presumption, we suggest that this is a good basis for a viable GGF binary timestamp format.

Summary

It has been observed that there is potential for multi-second errors in the interpretation of GGF timestamps. Here we have examined these issues and explored a possible solution. Our solution comprises functions to convert between ASCII and binary timestamps where leapsecond events are accounted for. The solution was designed to utilise the DataGrid R-GMA information and monitoring system, and has recently been integrated within R-GMA. In the next year we expect that it will be used widely within DataGrid and its applications. We also expect that the solution will scale well on a grid comprising many geographically dispersed sites and thousands of nodes.

References

[Sun 2001] Sun Microsystems, *JavaTM 2 Platform, Standard Edition, v1.2.2 API Specification*, http://java.sun.com/products/jdk/1.2/docs/api/index.html.

[Gunter and Tierney 2000] Gunter, D., and Tierney, B., *A Standard Timestamp for Grid Computing*, http://www-didc.lbl.gov/GridPerf/papers/StandardTimestamp.pdf, 22nd February 2000.

[Gunter 2001] Gunter, D., *private communication*, 2001.

[Gunter et al 2001] Gunter, D., Tierney, B. and Aydt, R., *A Timestamp Model for Grid Computing*, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-14-1.pdf, 28th June 2001.

[Gunter and Tierney 2001] Gunter, D., and Tierney, B., *An ASCII Timestamp Format for Grid Computing*, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-15-1.pdf, 28th June 2001.

[PerfWG 2001] Grid Performance Working Group, see http://www-didc.lbl.gov/GridPerf/

[NTP] Network Time Synchronization Project, see http://www.eecis.udel.edu/~ntp/

[USNO] Time Service Department, U.S. Naval Observatory, see http:/tycho.usno.navy.mil/

[USNO ftp] Time Service Department, U.S. Naval Observatory, see ftp://maia.usno.navy.mil/ser7/leapsec.dat

[Bernstein] Bernstein, D.J., A library for storing and manipulating dates and times. **see** http://cr.yp.to/libtai.html

[DataGrid] The DataGrid Project, http://www.eu-datagrid.org

[R-GMA] R-GMA, DataGrid WP3 - Information and Monitoring Services, **see** http://hepunx.rl.ac.uk/edg/wp3/