OpenVMS Technical Journal V15

New Time Features on OpenVMS and Read-Write Consistency Check Data Algorithm

Clarete Riana, Burns Fisher, and Sandeep Ramavana

New Time Features on OpenVMS and Read-Write Consistency Check Data Algorithm 1
Introduction 2
Read-write consistency check data algorithm 2
High-precision time 3
High-resolution time-since-boot 5
Method of ensuring consistency of UTC time across time zone changes 5

Introduction

With the increasing processing speed of newer processors and the growing transaction rates of certain applications, it becomes necessary for time to be tracked at a finer granularity than is currently available. To this end, we have added new time features on OpenVMS. They are as follows:

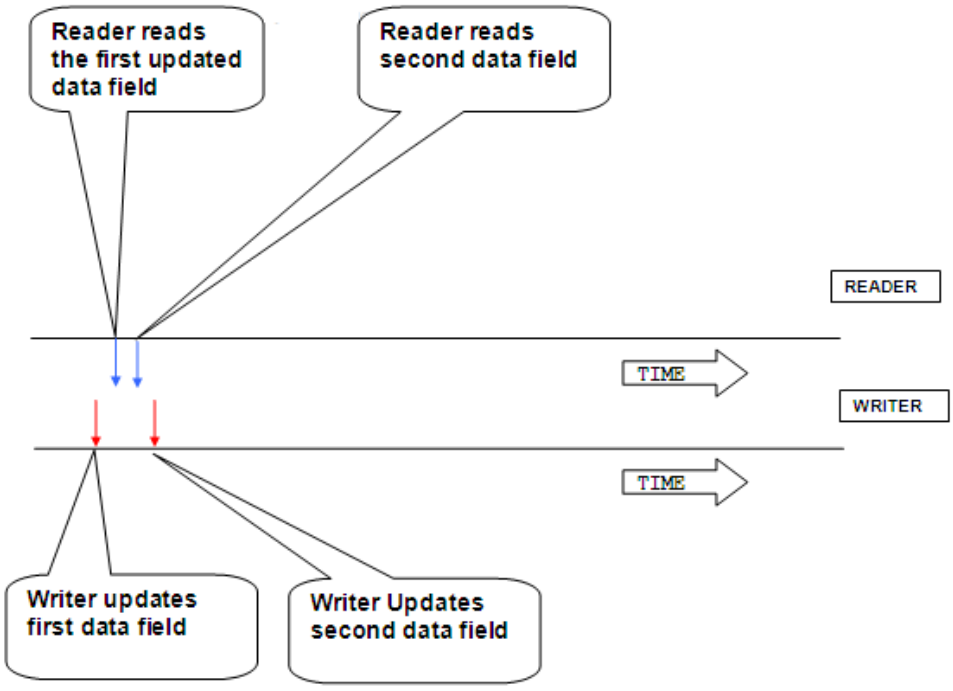
- A mechanism to obtain high-precision time using a new system service
- A mechanism to obtain a medium resolution time-since-boot of the system, by making use of a new flag with the existing $GETTIM system service.
- A method of ensuring the consistency of UTC time across a time-zone or summer/winter time change. This can be used by the applications to determine the right pair of TDF and system time, which otherwise can be a problem when seasonal time changes occur.
  This feature is also used by the $GETUTC service to ensure that $GETUTC never receives a glitch. The high precision time and consistent UTC change make use of a lightweight read-write consistency algorithm.

This paper describes each of the new features in detail.

Read-write consistency check data algorithm

When there are lockless concurrent accesses to shared resources, there will be scenarios where the reads of these fields will result in reading inappropriate values, as a write operation on these fields might be in progress.

The problem is captured in the figure below, where the writer updates the values of data fields one after the other and the reader reads the data fields. There is a possibility that the reader reads the data fields only when one of them is updated by the writer, and not the other, thereby reading the updated value of one field and old value of the other field, when it is required to read the updated values of both the fields.



There are existing locking mechanisms through which read/write consistency can be achieved to solve the above stated problem. Achieving read/write consistency of concurrent reads and writes across these data fields using the traditional locking mechanisms may prove very expensive in certain scenarios where write or read access must be fast. In such scenarios, we propose to make use of the unique lightweight checksum algorithm to achieve consistency of read/write of data fields.

When multiple data fields are to be written and read concurrently, in order to ensure read/write consistency, we propose to use another data field, which will contain appropriate checksum of the data fields. This checksum will be calculated and stored in the new data field during the write operation of the data fields whose read/write consistency is to be ensured. When there is an attempt to read the data fields, their checksum is again calculated using the same algorithm that was used during the write operation and matched with the checksum that was saved when the data fields were written. The reads are considered valid only if the checksum calculated after read matches the checksum stored during the write operation of the data fields. The field of checksum identification is well researched and various checksum algorithms are available. A checksum algorithm uniquely representing the data field is to be chosen here, which should detect if any of the data fields are changed.

It is to be noted here that this algorithm does not ensure any synchronization, but only ensures read/write consistency across multiple data fields. In other words, the algorithm does not ensure that the reader reads the updated values; it only ensures that it reads consistent values (perhaps the old values, perhaps the new ones).

High-precision time

Time services are provided by the operating system (OS) to give delta and absolute time services, including obtaining the time as well as alarm services at specified times. Modern day processors have timer clock hardware built into the chip itself. The timer clock hardware generates hardware interrupts based on a minimum time period, in clock ticks, specified by the OS. The timing generated from the timer interrupt is used to update system time in a granularity defined by the OS. There are some applications that may require higher resolution timer services. It is non-optimal to reduce the time period of the timer interrupts, as this would create more interrupt load on the system.

OpenVMS provides a system service $GETTIM, which returns a 64-bit number present in EXE$GQ_SYSTIME, which represents the current time in 100ns intervals beyond the Smithsonian base date of 17-Nov-1858. However, this time is only updated approximately every millisecond as a part of the hardware interrupt servicing that happens every millisecond. Thus, although the granularity of the OpenVMS system time is 100ns, the user can only have the accuracy of one millisecond.

Itanium processors have two registers that help generate the timer interrupt. One is writable by the OS and holds the value at which the next timer interrupt is generated, usually called the match register. The second register ticks in the order of a few megahertz depending on the type and speed of the processor, also called the cycle counter/time counter. When the value of this register matches that of the first register, a timer interrupt is generated, which updates the system time. The cycle counter and the match registers are utilized to obtain high resolution time.

A new system service $GETTIM_PREC is implemented on OpenVMS. The usage of $GETTIM_PREC system service is the same as that of the existing $GETTIM system service. The user has to pass a buffer in which high-precision time is returned. When the $GETTIM_PREC system service is issued--the system time updated at the last 1ms interrupt – the value of match register at the last 1ms interrupt and the current value of the cycle counter register are read.

While using these values to calculate high-resolution time, there should be effective consistency of the reads and writes of system time and match register values. One of the problem scenarios is when one of the processors issues a system call to get the high precision time, which in turn reads the global data cells, when another processor is in the interrupt service routine (ISR) updating the global data cells. This might result in reading one of the updated data cells but not the other. To solve this issue, we use the proposed method of ensuring read/write consistency. We save the checksum of the match register and system time in the ISR. So when the system call is issued, the saved system time, match register, and checksum are read and then the checksum of system time and match register is calculated locally. If the locally calculated checksum does not match the checksum that is read, it means that the system call was issued when the other processor was updating the data fields. So, the system call discards the values read, and re-reads them. Therefore, the data fields are re-read until the checksums are equal before calculating the high-precision time, thus making sure that the right pair of system time and match register are read. The checksum algorithm we used was based on selecting appropriate bit fields from system time and the time at which the interrupt occurred. The bit fields were selected such that the wrap time of checksum was a significant value.

This provides a higher resolution of system time than what is already available with $GETTIM system service. Further, since the high resolution time is based on the normal system time, it will be updated with $SET TIME, automatic seasonal time changes, or time synchronization services such as NTP.

This feature is an Itanium-only feature. The $GETTIM_PREC system service, when executed on ALPHA is equivalent to $GETTIM and just returns low-precision time in the user provided buffer, with a return status of SS$_LOWPREC, to indicate that only low-precision time is returned. On rare occasions, if the precision time calculation is inconsistent, you may also get a status of SS$_LOWPREC on Integrity systems.

Following is the output of executing the two system services in a test program, “High prec time.” In the output is the time returned by the $GETTIM_PREC system service, and “low prec time” in the output is the time returned by the $GETTIM system service. It is observed that $GETTIM_PREC returns a higher precision time value.

```
$ run diff_services
High prec time is A797CAAC6C88F6,  low prec time is A797CAAC6C7A08
High prec time is A797CAAC6C9C77,  low prec time is A797CAAC6C7A08
$
$ run diff_services
High prec time is A797CAAE2A6CB2,  low prec time is A797CAAE2A56A8
High prec time is A797CAAE2A8BFD,  low prec time is A797CAAE2A56A8
$
$ run diff_services
High prec time is A797CAB04E87BA,  low prec time is A797CAB04E6F28
High prec time is A797CAB04EA6FE,  low prec time is A797CAB04E6F28
$
$ run diff_services
High prec time is A797CAB11CCBAE,  low prec time is A797CAB11CBBA8
High prec time is A797CAB11CED45,  low prec time is A797CAB11CBBA8
```

As of today, there is no service that converts the system time returned to a high-precision ASCII time format. Conversion has to be done manually.

This feature is available on OpenVMS V8.3-1H1 with a patch kit and with all later OpenVMS releases.

High-resolution time-since-boot

Currently, OpenVMS has two data cells that are updated with the time-since-boot for the system. The first one, EXE$GQ_ABSTIM, is updated every second and therefore has an accuracy of a second. The second data cell, EXE$GQ_ABSTIM_TICS, is updated every 10 milliseconds, and therefore the accuracy of EXE$GQ_ABSTIM_TICS is only 10 milliseconds.

A new feature is added to provide a high-resolution time_since_boot. A new global data cell EXE$GQ_ABSTIM_100NS has been added to provide a high-resolution time_since_boot. This data cell is updated approximately every millisecond as a part of the hardware-interrupt servicing that happens every millisecond. Therefore, the time-since-boot of the system in this cell will have the accuracy of one millisecond, and a granularity of 100ns, equivalent to that of the system time in EXE$GQ_SYSTIME.

The user can read the high resolution time-since-boot directly from the global symbol EXE$GQ_ABSTIM_100NS, or obtain the high resolution time-since-boot using the existing $GETTIM system service. An optional new parameter flag has been added to the $GETTIM system service. When the system service is issued with this flag value as one, the $GETTIM system service reads and returns to the user the value in EXE$GQ_ABSTIM_100NS.

This feature is available on supported OpenVMS versions beginning with V7.3-2 (with a patch kit) and with all later OpenVMS releases on Alpha and Itanium.

Method of ensuring consistency of UTC time across time zone changes

Along with the global cell EXE$GQ_SYSTIME, OpenVMS has another data cell, EXE$GQ_TDF, which maintains the time differential factor. During seasonal time change, both data cells are updated to reflect the seasonal time change. There is a timing window present in updating the time data cells, EXE$GQ_SYSTIME and EXE$GQ_TDF. Reading the data cells in this window results in wrong values being read. We have seen this problem manifest itself in two cases. The first case is a glitch seen when $GETUTC was called around the seasonal time change. The second case is a Pthread library hang.

As a fix for this issue, a new data cell, EXE$GQ_UTC, is added, which is updated with the difference of EXE$GQ_SYSTIME and EXE$GQ_TDF as a part of the hardware-clock interrupt servicing that happens every millisecond.

$GETUTC simply reads EXE$GQ_UTC rather than calculating UTC from SYSTIME and TDF. EXE$GQ_UTC is always consistent because it is only updated while holding the HWCLOCK spinlock in the interrupt routine, as well as when the TDF is changed.

Pthread library code now reads the values of EXEGQ_UTC, EXEGQ_SYSTIME, and EXE$GQ_TDF, and compares them for consistency. The read-write consistency algorithm is used here again in this case to ensure that the right pair of system time and TDF is read. Library code locally calculates the UTC as a difference of EXE$GQ_SYSTIME and EXE$GQ_TDF, and reads the values in EXEGQ_UTC, EXEGQ_SYSTIME, and EXE$GQ_TDF in a loop until the locally calculated UTC matches the EXE$GQ_ UTC.