Continuous Clock Synchronization for Accurate Performance Measurements

Johannes Ziegenblag
Technische Universität Dresden
johannes.ziegenblag@tu-dresden.de

Matthias Weber
Technische Universität Dresden
matthias.weber@tu-dresden.de

ABSTRACT

Clock synchronization is a key prerequisite for accurate performance analysis. This is particularly true for most HPC systems, due to the lack of a system-wide timer. The effects of different clock properties lead to false measurements and wrong conclusions. Often these errors remain undetected by the user. Different environmental effects cause continuous changes to clock properties over time. This behavior is ignored by the usual post-mortem approach for clock synchronization. In order to improve time measurement accuracy we implemented a method for continuous clock synchronization. In this work we share our experiences and draw conclusions for tool development.

KEYWORDS

performance measurement, online clock synchronization, performance analysis

ACM Reference Format:
Johannes Ziegenblag and Matthias Weber. 2017. Continuous Clock Synchronization for Accurate Performance Measurements. In *Proceedings of Supercomputing 2017, Denver, Colorado, USA, November 12 2017 (SC17),* 2 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Driven by the need for more computational power, HPC systems become increasingly complex. To efficiently exploit their full potential performance analysis has become mandatory in the software development process. The analysis of parallel applications depends on accurate timing measurements in order to arrange application activities in the correct chronological order. This is important since application activities usually originate from a large number of processes. In many HPC systems each process measures time using its own clock, which usually leads to inaccurate results, if the clocks are not synchronized. Process-local clocks, although of the same type, may have different starting times, run at slightly different paces, or even accelerate over time. The reason is that clocks in HPC systems suffer from a range of environmental effect that influence the emitted timestamps:

Timestamp Jumps. Even though every clock provides the most basic and fundamental property—emitting time in a monotonic increasing way—some clocks may be prone to certain exceptions to this rule (e.g. the administrator changes the system time). Clocks may also be influenced by NTP corrections, and thus, accelerate or decelerate over time.

Timestamp Differences. Clocks may exhibit large differences between simultaneously measured timestamps. This results from different starting points of the clocks (e.g. the difference in boot time of two nodes can be days or weeks). Measured in nanoseconds ($10^{-9}s$) this number can be considerably large.

Differences in Speed. Clocks usually show differences in their pace, subsequently causing their timestamps to drift apart. Even if these differences are in the range of nanoseconds, with a sufficiently high resolution, their drift is still measurable. This is particularly true for clocks that are unaffected by NTP updates. Such clocks are designed to run at a constant pace and are prone to drift apart. Consequently, this behavior will lead to large timestamp differences over time.

Acceleration. Acceleration of clocks is a challenge even when the selected timer is not influenced by NTP updates. Acceleration or deceleration is the result of physical effects due to fluctuations of temperature or voltage. Moreover, aging related effects of clocks also contribute to these changes. In case of NTP influenced clocks acceleration effects occur more often and are therefore much harder to compensate.

The described effects may alter clock behavior only in the nanoseconds range, but are still measurable and accumulating over time. Since we cannot replace the process-local clocks with one global clock, we have to compute one logical global clock and translate all process-local timestamps accordingly. This one global clock allows to arrange events accurately, and thus, enables users to draw correct conclusions.

One-time synchronization or post-mortem approaches [2] are insufficient as clock behavior changes may vary over time. We implemented a continuous time synchronization method that defines one global time and synchronizes all other clocks to that global time. In order to scalably synchronize large numbers of clocks arrange them in a tree-like hierarchy. By employing a convex hull approach to determine the required correction factors, we achieve a high accuracy in the range of milliseconds. The error remains constant for all parallel process-local clocks even throughout several hours of application runtime.

SC17, November 12 2017, Denver, Colorado, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Synchronization of multiple clocks is required in many applications [2] also outside of HPC [6]. Consequently, numerous works on the matter exist. Zhang et. al. [7] and Ciuffoletti and Antonova [1] describe a synchronization method based on offset measurements between two clocks. Harrison and Newman [3], and Poirier et al. [5] employ and refine their approach. Zhang and Ciuffoleti describe a single convex hull approach, while Poirier and Harrison base their approach on two convex hulls. Jones and Koenig [4] developed a very accurate method of post-mortem clock synchronization. We base our prototype on the methods described in the mentioned works. While the basic principles are equal, we apply and evaluate the methods in the context of fine-grain performance measurements for large scale HPC applications.

3 CONTINUOUS SYNCHRONIZATION OF TWO CLOCKS

The basic idea for synchronizing two clocks is to determine the required correction factors (intercept and slope) for adjusting the local time of a client process to a global synchronized time. Therefore, the client process uses query messages to compare its timestamps with the timestamps of the global clock. For each message direction—request and receive messages—the current timestamps for sending and receiving the message are recoded and sent to the client process. The difference between both clocks (*offset* or *intercept*) is directly visible in the difference between both related timestamps. In the ideal case both clocks run synchronously. This would yield an offset of 0, i.e., send and receive timestamps are equal. However, this approach inherently includes the transfer time of the injected query message. This time cannot be distinguished from the offset between both clocks. The message transfer time is highly sensible to the current network conditions. Therefore only the lowest latency (smallest transfer time) messages provide a good value for the current offset between the clocks.

As clocks can accelerate/decelerate at any time, the offset can also change continuously. Thus, continuous query messages are necessary to capture changes of the offset. However, as only low latency messages provide good values for the offset, it is necessary to extrapolate the offset value during phases of higher latency messages. This extrapolation requires the computation of the current drift (*slope*) between both clocks. The larger the drift, the faster the times of two clocks run apart. Knowing the drift allows to extrapolate the offset from the last known low latency message to the current position.

Due to the large amount of values a linear regression to compute the drift is not feasible. A convex hull approach is applied to select only low latency messages (providing the highest accuracy) for the computation. A convex hull is computed for each message direction (request and response). The tangents between both convex hulls defines the drift value.

Intercept (offset) and slope (drift) are computed based on a sliding window technique. The width of the window controls the sensitivity/stability of the synchronization approach to clock changes.

4 MULTI-STAGE SYNCHRONIZATION OF MULTIPLE CLOCKS

To synchronize multiple clocks the pairwise approach can be extended. One option is to use a master-slave setup and let all clocks synchronize their local time with one clock defining the global time. However, this approach is not scalable and may lead to overloading of the process with the global time. A more suitable option is to follow a multi-stage approach. It is possible to arrange clock pairs in a tree-like hierarchical fashion. The global time is defined by the clock at the root level of the tree. Clock pairs synchronize their local times with clocks at the above tree layer. This way the global time is distributed downwards throughout the tree hierarchy. This setup balances the overall workload and allows to scalably synchronize large number of clocks.

5 CONCLUSION

Our experiments have shown that clocks on HPC systems exhibit a measurable offset, noticeably impacting performance analysis. This necessitates clock synchronization for accurate performance measurements. Besides large timestamp differences at application startup, also time differences due to clock drifts can amount up to more than hundreds of milliseconds in an hour. As clock properties are changing over time, continuous synchronization approaches are required for long running applications. Our approach improves measurement accuracy especially for such application scenarios. Our prototype implementation provides accurate timings in the range of milliseconds even for several hours of application runtime.

In our experiments we found that a sliding window of 5 minutes length provides the best trade-off between stability and sensitivity to clock changes. We recommend the usage of monotonic-raw-timers, since they are unaffected by NTP updates and provide the best overall stability.

We extend upon previous work by employing a hierarchical synchronization scheme that can compute correction factors for a large number of clocks with acceptable overhead. Our work has shown that time synchronization methods based on intercept and slope computation are applicable in the context of performance measurements of parallel applications.

REFERENCES

[1] Augusto Ciuffoletti and Galina Antonova. 2005. Clock Phase Change Compensation Using Graham Scan. In *Conference on IEEE-1588 Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems.*

[2] Jens Doleschal, Andreas Knüpfer, Matthias S. Müller, and Wolfgang E. Nagel. 2008. Internal Timer Synchronization for Parallel Event Tracing. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface.* 202–209.

[3] A. Harrison and P. Newman. 2011. TICSync: Knowing When Things Happened. In *2011 IEEE International Conference on Robotics and Automation (ICRA).* 356–363.

[4] Terry Jones and Gregory A. Koenig. 2010. A Clock Synchronization Strategy for Minimizing Clock Variance at Runtime in High-end Computing Environments. In *Proc. of the 22nd Intl. Symposium on Computer Architecture and High Performance Computing.* 207–214.

[5] Benjamin Poirier, Robert Roy, and Michel Dagenais. 2010. Accurate Offline Synchronization of Distributed Traces Using Kernel-Level Events. *SIGOPS Oper. Syst. Rev.* (2010).

[6] L. Schenato and G. Gamba. 2007. A distributed consensus protocol for clock synchronization in wireless sensor network. In *2007 46th IEEE Conference on Decision and Control.* 2289–2294.

[7] Li Zhang, Zhen Liu, and C. Honghui Xia. 2002. Clock Synchronization Algorithms for Network Measurements. In *IEEE INFOCOM 2002. Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies.*