An evaluation of the state of time synchronization on leadership class supercomputers

Terry Jones1 | George Ostrouchov1 | Gregory A. Koenig1 | Oscar H. Mondragon2 | Patrick G. Bridges3

1Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge TN, USA
2Universidad Autonoma de Occidente, Cali, Colombia
3Department of Computer Science, University of New Mexico, Albuquerque NM, USA

Correspondence
Terry Jones, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA.
Email: trj@ornl.gov

Summary
We present a detailed examination of time agreement characteristics for nodes within extreme-scale parallel computers. Using a software tool we introduce in this paper, we quantify attributes of clock skew among nodes in three representative high-performance computers sited at three national laboratories. Our measurements detail the statistical properties of time agreement among nodes and how time agreement drifts over typical application execution durations. We discuss the implications of our measurements, why the current state of the field is inadequate, and propose strategies to address observed shortcomings.

KEYWORDS
clock synchronization, large-scale systems, system software, time service

1 INTRODUCTION
The trend towards increasing node counts in high-performance computing (HPC) is motivating a move toward greater levels of concurrency in HPC systems. Today’s software environment is now being called on to produce new solutions for emerging issues including managing system power, resilience, and performance characteristics. The distributed algorithms that underlie such services operate much more efficiently in the presence of tightly synchronized clocks. For example, tightly synchronized clocks benefit well-known gang scheduling techniques and complex consensus algorithms. To illustrate the point, such time synchronization enables more aggressive assumptions about communication and synchronization patterns, the removal of unnecessary locks, and a wide range of other applications. Clock-based techniques are already frequently deployed in cloud and data center distributed systems for precisely these reasons.

We examined the time synchronization on some of the world’s fastest and most powerful machines. These leadership-class systems employ high-end hardware connected by an extremely low-latency, low-jitter, interconnect in a carefully controlled environment, in contrast to widely distributed cloud-based systems based on commodity hardware and networks. Because of this, we assumed that these systems would have more stable, predictable hardware clocks, and close base time agreement using only standard time synchronization systems like Network Time Protocol (NTP). We did not believe that the complex hardware and software techniques used to provide time synchronization in wide-area systems would be necessary in leadership systems.

Our results demonstrate that the actual time uncertainty for leadership-class machines is often unexpectedly large, in some cases over 600 milliseconds despite network latencies of less than two microseconds. Building on this, we set out to thoroughly quantify the magnitude of the time synchronization challenge in leadership-class systems. This study shows that the current time protocol in use, NTP, is not suitable for providing the level of time synchronization necessary for important system software tasks such as coordinated scheduling. Based on this, we conclude
that more complex time synchronization techniques are in fact needed to provide tight time synchronization in these systems, and we discuss the specific techniques most appropriate to HPC systems according to our findings.

### 1.1 Timing terminology

A perfect clock, denoted by \( t \), reports one second of elapsed time per second of real time and has an origin \( t = 0 \) at a specified previous instant. A given real clock is imperfect; it can have acceleration, temperature, and other environmental sensitivities. Using the notation found in Veitch et al.,\(^1\) a real clock reads \( C(t) \) at the true instant \( t \) and suffers from an error or offset which we denote as \( \theta(t) \):

\[
\theta(t) = C(t) - t
\]

at true time \( t \). The skew \( \gamma \) corresponds to the rate of change in clock offset:

\[
\gamma(t) = \lim_{h \to 0} \frac{\theta(t + h) - \theta(t)}{h}.
\]

Uncertainty \( \phi \) is the maximum offset between any pair of observed real clocks:

\[
\phi = \max_{i,j,t} |\theta_i(t) - \theta_j(t)|,
\]

for \( i, j \) in a set of clocks and \( t \) over a time interval under consideration. To synchronize is to coordinate the occurrence of multiple entities with respect to time. Two clocks are synchronized to a specified uncertainty if their measurements of the time of a single event at an arbitrary time differ by no more than that uncertainty.

NTP stands for Network Time Protocol, and it is an Internet protocol used to synchronize the clocks of computers to some time reference. NTP is an Internet standard protocol originally developed by Professor David L. Mills at the University of Delaware.\(^2\)

### 1.2 Modern HPC architecture

The basic building blocks of modern HPC machines are nodes connected via a communications interconnect. A node usually consists of a processor or multiprocessor, memory, an interface to the interconnect and, optionally, a local disk. Modern interconnects are capable of sustaining concurrent transfers measuring gigabytes per second between multiple nodes. The typical latency between the request for a transfer and the first bytes appearing at the destination are on the order of a few microseconds or less.

Figure 1 depicts a modern high-performance computer such as those found among the world’s most powerful machines.\(^3\) The machine features a large number of compute nodes that are able to pass information among each other via the interconnect. As their name would suggest, compute nodes are optimized for computation; they usually do not contain a local disk. For other aspects of computer activity (e.g., interactive user activity, disk drive accesses, and so forth), support is usually carried out by service nodes set aside for these necessary operations including login nodes and I/O nodes. For example, the Titan machine at Oak Ridge National Laboratory consists of 18,668 compute nodes and 20 service nodes that are used for interactive logins and access to the parallel file system.

The ability to efficiently handle timesharing within compute nodes has become a requirement for HPC machines. Traditionally, these supercomputers have operated in a manner designed to optimize the machine’s resources. As such, application teams submit production-ready versions of applications to a batch queue of jobs. Each job request contains requirements for the number of compute nodes and the expected overall execution time. When the job reaches the top of the queue, it is given sole access to a matching number of compute nodes. While no other user

![A typical high-performance computing architecture](image-url)
time-share with the user application, termed noise or detours in the literature, may lead to undesirable performance impacts on the user application if left unchecked. Efficient timesharing on compute nodes is very important whether the need arises from enabling colocated threads (e.g., in situ analytics or coupled applications), or from accommodating computational support for an ongoing experiment, or simply from supporting system software daemons efficiently.

Current generation supercomputing systems with relatively large node counts cause time synchronization issues to manifest in terms of a large number of processes all reporting skewed clocks as described and quantified in this paper. The issue of fine-granularity time synchronization is also relevant to next-generation supercomputing systems. In next-generation systems, system architectures are trending toward a relatively smaller count of nodes with each node containing a much higher degree of parallelism. The expectation is that the processes within a single node would likely all read consistent time from a single node-specific time source. Accordingly, it follows that across an entire next-generation supercomputer, large groups of processes would all agree on a value for the current time, with time skew observed between groups. The challenge is that a time perturbation on a single node can simultaneously affect a large number of processes participating in a computation while the opportunity is that getting the nodes into tight time synchronization with one another can quickly improve the time base for all processes in a computation.

Two more definitions will help to acquaint those unfamiliar with HPC to the basic environment. Message Passing Interface (MPI) is a standardized and portable message-passing system designed for the development of portable and scalable large-scale parallel applications. Finally, Bulk Synchronous Parallelism (BSP) algorithms are those that consist of concurrent execution of local computation across processes, and allow parallel progress through the use of synchronization points.

1.3 Contributions of this paper
To our knowledge, ours is the first paper that presents a detailed statistical analysis of clock agreement on high-performance computer environments. We introduce collected results from a number of tests conducted on some of the world’s most powerful computers. We then analyze these results and how they might apply to different applications and scenarios. Finally, we discuss the implications of our measurements and why the current state of the field is inadequate, and we propose strategies to address observed shortcomings. The data associated with this paper are being made available online; the registered DOI value is 10.11578/1130048.

At the outset of the work presented in this paper, we expected to find a much lower degree of uncertainty among node clocks in large-scale supercomputers than our results uncovered. We believe that the reason time synchronization has been neglected to this point is due to the way that supercomputers are used by the research community versus how they are provided as resources by supercomputing centers. System software researchers are possibly in the best position to be able to leverage the benefits of tight synchronization of node clocks but are unlikely to be in a position to affect a change in how a given supercomputer is run. On the other hand, the operations personnel who are responsible for maintaining supercomputer systems may not recognize the value of tightly synchronized node clocks or know the degree to which node clocks disagree, mistakingly believing that the temperature and power controlled environment in data centers along with the use of commodity protocols to synchronize time, such as NTP, is enough. Our aim with this paper is to highlight these ideas, supported by our measurements and findings, to the supercomputing community.

1.4 Paper outline
The rest of this paper is organized as follows. Section 2 provides a more complete motivation for our line of study including its significance and a quantitative assessment of the impact of time synchronization to parallel applications. Section 3 describes work related to our own. Section 4 provides a series of relevant data measurements, followed by a discussion of these measures in Section 5. Section VI presents recommendations stemming from our study. The paper closes with our future plans and a conclusion.

2 IMPORTANCE OF TIME AGREEMENT
The main motivation to this line of inquiry is efficiency, primarily in wall-clock runtime of parallel applications written in the bulk synchronous parallel (BSP) style. Efficiently employing concurrency is critical to success in HPC. With growth in clock frequency stalled, performance improvements are being achieved by an exponential increase in the number of processing elements per chip, an increase in the hardware threading per core, and an increase in the total node count. These trends underscore the importance of parallel speedup.

Other kinds of efficiency, such as in maintenance, machine uptime, and code development are impacted by time agreement, although these are more difficult to quantify. For example, resilience root cause analysis relies on good time agreement to inform log analysis when dependency is ambiguous. Performance analysis tools also rely on time agreement for effective distributed debugging. Finally, timed guarantees can be used to avoid some communication. Such clock-based techniques are already frequently deployed in cloud and data center distributed systems; higher-quality time synchronization would allow the use of these techniques in HPC systems. The negative consequences of high time uncertainty and the benefits of low time uncertainty are discussed in this section.
2.1 Improved clock synchronization significance

The lock-step progress of BSP style programs is bounded by the slowest task among the parallel task set. That is to say, the parallel application cannot advance until all tasks reach a common point. However, as node counts increase, the likelihood of one task experiencing an interruption becomes a key factor to performance; as the BSP per-loop iteration time decreases, the effects of even smaller noise perturbations increase, further emphasizing the usefulness of tightly synchronized clocks. To mitigate the potentially disastrous hit on performance, sophisticated runtime systems and operating systems seek to overlap any disruption in progress across all compute nodes. When system software* is able to schedule all interrupting tasks on every node such that they occur simultaneously, the cascading effect on the parallel algorithm is kept in check—a strategy known as coordinated scheduling.

A number of works have used coordinated scheduling-based techniques to minimize the impact of potentially interfering workloads. In this kind of solution, co-schedulers on each node ensure that the parallel tasks are scheduled at the same time. This can be achieved by coordinating interfering short-lived system tasks. Coordinated scheduling schemes face several technical challenges at various levels of scale. In order to support this, next architectural designs in HPC systems must provide low-overhead synchronization/coordination mechanisms. The scheduler must guarantee precise CPU reservations for cooperative codes and accurate timing to run applications with coordination needs across nodes. In this scenario, a precise inter-node time agreement is critical. Moreover, the absence of suitable time agreement prevents effective gang-scheduling and coordinated-scheduling.

2.2 How precise is enough? The impact of time synchronization to applications

Each parallel application has its own sensitivity to noise, and therefore its own capacity to be aided by the low-time variance strategy of coordinated scheduling. The extent of wall-clock speedup depends on how frequent synchronizing collectives (e.g., MPI_Barrier, MPI_Allreduce) are utilized within the application. On the upper bound, our earlier work demonstrated that a coordinated scheduling strategy can provide a speedup of 285% for sensitive bulk synchronous workloads.

Here, a sensitive application is one in which an MPI synchronizing collective call (e.g., MPI_Barrier, MPI_Allreduce, MPI_Allgather, and so on) is called at least once per iteration in an iterative simulation or algorithm. Previous works have shown that interference sources have potentially greater impact on applications with fine-grained parallelism (i.e., applications with shorter per-iteration intervals). For example, in Seelam et al., authors show that jitter can generate slowdowns as high as 8% for applications with computation intervals of 100 ms and over 16% of slowdowns for applications with 10-ms computation intervals at 32 K CPUs.

The amount of time agreement required for coordinated scheduling varies nonlinearly with time agreement and requires an uncertainty bound of several microseconds before attaining best results.

Figure 2 shows how the performance of different applications changes in relation to varying the degree of synchronization of the time-shared analytics. This figure, reproduced from Levy et al. work, presents analytics synchronization levels that range from the perfectly coordinated case to the completely uncoordinated case for 64 K nodes. Different levels of synchronization are simulated adding offsets to the time at which noise traces start on different analytics processes. In this work, authors show that the amount of synchronization needed to mitigate interference from system software and other workloads can vary depending on the characteristics of the application and interference source. For high-impact noise sources (e.g., asynchronous checkpointing or co-scheduled analytics), synchronization amounts between 20 and 100 ms are necessary to reduce the performance impact to most HPC applications below 10%. The same studies show that common application communication patterns can synchronize global activities within approximately 100 ms but not necessarily more tightly.

While scheduling is often critical to parallel algorithms and the impact of harmful interruptions or detours should be minimized, some BSP algorithms can be made to operate in a more asynchronous manner. In perhaps the largest impact reported, Hammouda et al. studied a classic bulk synchronous implementation of an explicit stencil calculation in a range of environments spanning the absence of random detours to increasingly larger presence of random detours. They demonstrated that by refactoring explicit stencil calculations, they were able to achieve speedups of 1 to 37 times faster than a traditional noise sensitive algorithm in an uncoordinated timing environment.

Beyond HPC, the impact of timing to more general scenarios in concurrency is also interesting. Google’s Spanner uses coordinated time to achieve scalability and guarantee correctness of its distributed concurrency features such as externally consistent transactions, lock-free read-only transactions, and non-blocking reads. Liskov showed how synchronized time is used for generating authentication tickets in Kerberos, for ensuring cache consistency in an object persistence system, for ensuring atomicity in a distributed database, and for improving the performance of commit window negotiation in distributed filesystems. Liskov points out that the use of synchronized clocks in a distributed system can improve performance by replacing communication with local computation.

---

* System software here refers to the various software that supports the execution of the application, including the OS kernel, daemons, and programming system runtime libraries that manage resource allocation and provide basic system services to the application.
2.3 The time agreement tradeoff: interruptions versus time agreement precision

Given the importance of time agreement to parallel applications, one might think that simply relying on a client which frequently restores a common time base among a distributed set of nodes would be a simple solution. Theoretically, a client design in which a subset of nodes with access to a reference clock could utilize a protocol with all other nodes to overcome the drift with each node’s local clock. Indeed, such clients exist, but they frequently fail to provide the desired benefit due to a subtle design tradeoff: most client designs require frequent updates among all nodes to reach very high levels of time agreement, but frequent updates result in increased system software interrupts to carry out the clients message traffic. On the one hand, the advantages of very precise time agreement (i.e., the clocks on all nodes are coordinated to agree to within 1 microsecond) are very attractive to many parallel uses including consensus algorithms and parallel tools. On the other hand, the clients which enable precise time agreement may also negatively impact common operations such as synchronizing collectives through their heavy use of software interrupts.

Ferreira et al. studied the impact of harmful system software interrupts or detours to BSP applications. They demonstrated that a 2.5% net processor noise rate (interruption rate) at 10,000 nodes can have no impact or can result in over a factor of 20 slowdown for the same application, depending solely on how the noise is generated. Their study featured important DOE scientific applications including the CTH shock physics simulation, the POP parallel ocean simulation program, and SAGE which is SAIC’s adaptive grid Eulerian hydrocode. Such noise rates would require a bounded time uncertainty of less than 5 microseconds.

Hoefler et al. showed that the scale at which noise becomes a bottleneck for three applications: SWEEP3D, AMG, and POP. SWEEP3D solves a neutron transport problem on a 3D Cartesian geometry and AMG is an algebraic multigrid solver. They found that the impact of uncoordinated interrupts at 16,000 processes ranges from 4% slowdown (AMG) to over 100% slowdown (SWEEP3D and POP). To avoid these slowdowns, a time uncertainty bound of several microseconds would be required.

In our measurements described below, the time agreement clients on each distributed node were configured to perform agreement consensus every few hundred seconds with the hope that any localized drift during the period between updates would be inconsequential. As we show below, the time agreement for this strategy was surprisingly poor.

2.4 Relevant questions related to time agreement in HPC environments

A detailed understanding of the degree and characteristics of clock-agreement are useful for the above lines of inquiry. Moreover, environments that include parallelism, distributed agents, or coordination may be concerned with subtle aspects of clock synchronization including

- What does the distribution of offsets from a reference clock look like? What are the statistical properties?
- What is the magnitude of time uncertainty across a set of clocks?
- How can we reason about clock skew and synchronization over short and long durations?
- What do these results mean for various applications?
3 RELATED WORK

Clock synchronization in distributed systems is an important problem that impacts design decisions in a wide range of applications. To that end, clock synchronization has been well surveyed and has a long history with many important contributions.\textsuperscript{24–27} This is in keeping with the wide range of usage models that benefit from synchronized clocks. For example, software updates rely on accurate file time stamps across distributed nodes to support a wide range of activities from system administration configuration tracking to software development team code repository tracking. These file time stamp usage cases typically depend on agreement within a few seconds. Parallel environments like those found in high-end computing have much more strict requirements. For example, application development tools like message trace analysis tools can show problematic message-traffic patterns, but their effectiveness is undermined without tight enough constraints on time variance between nodes.\textsuperscript{28} Another important potential use for high performance distributed clocks is found in the general area of system software. For example, file system tokens used for flow control and metadata management benefit from skews held to a few dozen microseconds or less.\textsuperscript{29}

Perhaps one of the best known clock synchronization systems is NTP\textsuperscript{2} NTP uses a hierarchical system of levels of clocks, with the highest level ("Stratum 0") considered as the reference and in practice typically being atomic clocks or GPS clocks. A node in the network usually synchronizes its local clock against one or more clocks in the same stratum or the stratum one level higher. If the remote clocks are accessed over the public Internet, the node can usually maintain its clock to within ten milliseconds of the reference clock; accuracies of one millisecond are possible if a time server is available on the local intranet.\textsuperscript{30} Hierarchical approaches trade scalability for relaxed tolerance to variance: each leaf node is no longer necessarily synchronized with the stratum 0 reference, and there is no guarantee that the stratum 0 reference is exactly in synch with lower level stratum (only the NTP protocol itself). Mills provides a survey of the characteristics of roughly 10,000 machines on the Internet.\textsuperscript{31} Offset error distributions for several protocols show results ranging from multiple milliseconds to times exceeding 100 seconds.

More recently, Veitch et al. described RADclock, an improved clock synchronization system for distributed clocks.\textsuperscript{1,32} Where NTP looks behind and slowly moves skew based on a reference clock, RADclock’s feed-forward design looks ahead and rapidly removes skew based on a least-squares fit to projected skew. Both NTP and RADclock ensure time remains monotonically increasing on all controlled clocks (time is never adjusted backward). RADclock’s performance is dependent upon jitter in the communication paths and has demonstrated skew of less than 50 microseconds over the course of 5 hours with Ethernet connected PCs. Unlike NTP, RADclock requires a Linux kernel patch.\textsuperscript{33}

Precision Time Protocol,\textsuperscript{34} also known as IEEE 1588, is a standardized protocol designed to provide a precise clock for test and measurement systems. PTP is typically used in conjunction with hardware support and is applicable for embedded systems and military applications. A software-only implementation of PTP has been developed by Correll et al. for Test and Measurement Systems;\textsuperscript{35} however, it does assume a hardware-supported master clock. The idea of PTP is to utilize additional time stamps to remove delay uncertainties and thereby improve time agreement.

Hong et al. described ClockScalpel, a tool for measuring clock time accuracy capabilities among distributed clock synchronization protocols.\textsuperscript{36} They utilized a specially equipped test beds with a GPS receiver to measure delay characteristics. Their findings among 1861 public time servers showed synchronization protocols were able to achieve a median error of 2 to 5 milliseconds but with a long tail.

Google’s Spanner\textsuperscript{21} is a scalable, globally distributed database that directly addresses challenges involving scalability and consistency of trillions of rows of data that span multiple data centers. An API called TrueTime provides each participating process access to a time interval ($T_{\text{earliest}}$, $T_{\text{latest}}$) with a bounded uncertainty and an absolute time that is guaranteed to fall within the interval. The uncertainty increases between time synchronizations, which the authors report as happening at approximately 30-second intervals. The resulting worst-case time uncertainty is in the range of 1 to 7 milliseconds (drift rate of approximately 200 microseconds/second). Spanner synchronizes time via GPS and atomic clock time references.

In Liskov,\textsuperscript{22} Liskov provides a survey of applications of synchronized clocks in distributed systems. A key observation is that clock synchronization cannot be provided absolutely, only with some high probability of falling within a bounded range. Our work presented in this paper provides empirical measurements for clock skew bounds, and we believe that this is important toward developing an understanding of the types of applications that are possible.

In Maillet and Tron,\textsuperscript{37} the authors present statistical methods to estimate the relationship between the values of different clocks. Two algorithms proposed in the literature are analyzed; the sample data needed by the estimators are described and the qualities of the estimators are compared. They note a tradeoff in the design of statistical clock synchronization methods: a long interval of data collection is useful for reducing the impact of noise in the delay measurements, but the accompanying overhead is undesirable to other workloads which may be impacted by the delay to achieve time synchronization. To address this issue, the authors propose using a sample before and after methodology when possible.

In Gurewitz et al.\textsuperscript{38} by Gurewitz, Cidon and Sidi, a more complex peer-to-peer approach for time synchronization is presented. The approach, referred to as Classless Time Protocol or simply CTP is non-hierarchical and based on peer node exchanges. The CTP design explicitly tunes to minimize a global network-wide cost function. The authors assume an environment with significant differences than our intended environment: (1) CTP assumes a generalized network with no assumptions on uniformity of links (other than link delays cannot be negative); (2) they assume no clock skew among non-master clocks, although they reference techniques that may remove skew. With these underlying premises, CTP uses minimum one-direction delays and an assumption that clock offsets cancel in circular paths. The authors of CTP state in Gurewitz et al.\textsuperscript{39} that one-way delay is preferable because round-trip delays may be subject to route differences between outbound and return. Our approach differs with these generalized approaches in several key ways. First, in our unique environments, we are only concerned with symmetric networks within a supercomputer and dedicated hardware. Unlike a generalized network, we have guarantees on routing, symmetry, and workload which leads to round-trip times that are extremely consistent and three orders of magnitude or more faster than arbitrary networks. The magnitude of round-trip
times in our network environment is orders of magnitude below the clock offsets so that the additional precision potentially gained with the CTP approach (that would still need a skew removal technique) would not change our conclusions.

Generalized network strategies like CTP utilize statistical estimation techniques to determine the message delay. When the variable delay in message delivery is Gaussian distributed, the optimal parameter estimator is relatively easy to derive. However, when the variable delay is not Gaussian, a more sophisticated technique is required. In Jeske,40 Jeske addresses the delay characteristics for exponentially distributed delays. Jeske develops the maximum likelihood estimation (MLE) which features particularly attractive characteristics due to its asymptotic properties: it is unbiased and achieves the Cramer-Rao bound (CRB) for joint skew and offset estimation at large enough numbers of data samples.

In Doleshal et al.,41 Doleshal et al. introduce a two-step clock synchronization process suitable for program development tools such as a parallel event tracer. First, local times are captured from the independent clocks running on separate nodes of a large parallel machine. Second, a post mortem step is used after the completion of the application to reconcile the clock differences found with a trace log and restore the relationship of concurrent events. To minimize the impact the workload being traced, local clocks are accessed during operations which are performed on all nodes (thereby reducing the likelihood of imbalanced overhead). The authors are able to take advantage of certain constraints introduced in the environment to perform the post mortem step (e.g., a message must be sent before it is received). Using a strategy based on the aforementioned CFP method, the post mortem step is able to display events in their corrected order and relation.

In our previous work,42,43 we designed a clock synchronization protocol that yields excellent time agreement precision while controlling the deleterious effects of client interrupts. We were able to achieve agreement levels of a few microseconds. Our technique leverages the MPI profiling interface (PMPI) to intercept calls to collective operations in order to piggyback all necessary protocols. Our implementation focuses on MPI_Barrier() only, however the technique could be carried through all collective operations in the MPI library in a straightforward way. The synchronization operation must involve all processes in an MPI computation in a transparent manner that does not disrupt the normal functionality of the MPI library. Accordingly, our technique internally copies the MPI_COMM_WORLD communicator into a new MPI Communicator that includes all processes in the computation but is semantically independent of the MPI_COMM_WORLD communicator; This new communicator is not visible to the MPI library user. After every $N$ collective operations that involve all processes in the computation, the MPI library performs a clock synchronization collective involving all processes. As a performance optimization, the library does this clock synchronization collective after $N$ collective operations and after a minimum specified wall-clock time has elapsed since the last time synchronization collective under the notion that node clocks are unlikely to fall out of synchronization soon after the last synchronization operation. Overall, our approach is suitable for MPI parallel applications but does not address time agreement outside of an MPI context.

The work we present in this paper differs from the Related Work in two ways. First, the work presented here is not a mechanism for synchronizing node clocks such as NTP, RADclock, or Precision Time Protocol. Second, the work presented here is not a piece of system software that leverages tightly synchronized node clocks such as Spanner or the work presented in Liskov’s survey. Instead, the work presented here is the technique and measurements for node clock synchronization in supercomputers.

## 4 | IMPLEMENTATION DETAILS AND DATA MEASUREMENTS

In this section, we describe the test environment, our method for collecting the measurements, and characterize our measurements.

### 4.1 | Test environment

We conducted our experiments in three different supercomputers: Titan, Edison, and Mira. Three different interconnects were explored: Titan uses a Cray Gemini interconnect, Edison uses a Cray Aries interconnect, and Mira uses a BlueGene/Q interconnect (see Figure 3).

![Interconnects topologies](image_url)
4.1 Titan
Titan is located at Oak Ridge National Laboratory. A Cray XK7 features a hybrid-architecture with a peak performance of 27 Petaflops. It contains both 16-core AMD Opteron CPUs and NVIDIA Kepler GPUs. Titan incorporates 18,688 compute nodes and 710 TB of memory.

Titan is based on Cray’s Gemini interconnect with a three-dimensional torus topology (see Figure 3). This 3D torus has the dimensions of Z=24, X=25, Y=16 (that is, 24 blades per cabinet, 25 cabinets, 2*8 rows). In Gemini, near-neighbors are given more bandwidth than farther clients, which receive geometrically less a share of bandwidth according to fan-in/distance. This means that for large machines, unless each link has adequate bandwidth for the traffic pattern, the chances of contention increase as messages travel farther and farther; large fan-in communications therefore can pose significant challenges for such topologies. End point and per-hop latencies for Gemini are presented in Table 1.

4.1.2 Edison
Edison is a 5576-node Cray XC30 cluster with a total of 357 TB of memory. It contains an Intel® Xeon® E5-2695 processor (12-core Ivy Bridge). The Eos compute partition is comprised of 133,824 processor cores. The Dragonfly topology interconnect of Edison is configured with up to 240 Blue links using 60 optical cables, 4 links per cable, between each cabinet; there are fifteen two-cabinet groups.

Edison uses Cray’s Dragonfly topology. Systems can be configured to meet bandwidth requirements by varying the number of optical connections. Bidirectional bandwidth for two Aries nodes at 4-K message size is approximately 14.3 GBytes/s. Peak global bandwidth is 11.7 GBytes/s per node for a full network; with a payload efficiency of 64% percent this equates to 7.5 GBytes/s per direction. This topology enables a higher bandwidth and reduced latency in comparison with Gemini. Table 1 shows the end point and per-hop latencies for Aries.

4.1.3 Mira
Mira supercomputer is a 49152-node IBM Blue Gene/Q system located at Argonne National Laboratory. This machine has a total of 786,432 cores, 786 TB of memory and a peak flop rate of 10 PetaFlops. Each node is outfitted with 16 GB RAM memory and a 1.6-Ghz PowerPC A2 processor with 16 cores. The whole system is accommodated in 48 cabinets.

Mira uses a 5D Torus proprietary interconnect. Each node has 10 chip-to-chip links with a bidirectional bandwidth of 4 GB/s. Each node board contains 32 compute cards. A midplane connector connects boards in the same midplane. Midplanes are interconnected through fiber optics cables at 10 Gbps. A rack contains up to two midplanes. The torus interconnecting midplanes comprise 512 nodes and has a size of 4 x 4 x 4 x 2. This 5D topology enables a higher nearest-neighbor bandwidth and reduced latencies across the network. Morozov et al. in Morozov et al. show that latency in Mira is not significantly affected by the distance between nodes. Table 1 shows the end-to-end and per-hop latencies for directly connected nodes and for nodes separated by the longest distance (96-rack system, 31 hops).

4.2 Timeline: a tool for evaluating clock agreement
To carry out the experiments described in this paper, we developed Timeline, an MPI program designed for HPC machines with low-latency interconnects. Timeline is capable of high accuracy time agreement measurements among the members of an MPI communicator. First, a reference node is located. For simplicity, this reference node is chosen to be Rank 0 in the MPI computation, although any node within the computation would be a valid candidate. The algorithm measures the time agreement offset, \( \delta \), between the reference node’s time and each other node’s time.

Figure 4 depicts a timeline view of how the clock offset measurement process takes place between two processing elements. In the figure, time advances from top to bottom. The reference node acts as a master to all other nodes in a master:slave relationship as shown in Figure 4. The master process begins by exchanging one message with one of the slave processes. The master process initiates the message at time \( t_{m1} \) on the master’s clock and the slave process responds to the message at time \( t_{s1} \) on the slave’s clock. Time measurements from this initial message exchange are discarded because they are likely to include one-time anomalous overheads. Next, ten sets of messages are exchanged between the master process and the slave process, and these results are recorded in a report block. In each set, the first message is initiated at time \( t_{m1} \) on the master’s clock, the response to this message is sent at time \( t_{s1} \) on the slave’s clock, and the response is received a time \( t_{m2} \) on the master’s clock. The resulting tuple for set \( i \), \( \langle t_{m1}, t_{s1}, t_{m2} \rangle \), is recorded in the experiment’s report block. After recording ten tuples for the slave, we determine the tuple with minimum round-trip time, \( t_{m2} - t_{m1} \). We denote this tuple by \( \langle t_{m1}, t_{s1}, t_{m2} \rangle \), dropping the index \( i \), and output as our measurement for the slave. Finally, this entire process is repeated for all slaves in the computation.

| Architecture          | Available latency measurements |
|-----------------------|--------------------------------|
| Cray XK7 Gemini       | 8-byte PUT 0.7 \( \mu s \)     |
|                       | Small MPI message 1.5 \( \mu s \) |
|                       | Per-hop latency 105 ns         |
| Cray XC30 Aries       | 8-byte PUT 1.6 \( \mu s \)     |
|                       | 8-byte MPI message 1.3 \( \mu s \) |
|                       | Per-hop latency 100 ns         |
| BlueGene/Q            | Direct neighbor 0.3 \( \mu s \) |
|                       | Farthest neighbor 2.5 \( \mu s \) |
|                       | Per-hop latency 35 ns          |

Table 1: Latencies for Cray XK7 Gemini, Cray XC30 Aries, and BlueGene/Q
FIGURE 4 Timeline view of message sends and receives for the experiment

From a measurement of the form \(< t_{m1}, t_{s1}, t_{m2} >\), we define the following three quantities for use in further discussion, analysis, and figures:

- **trip** as \(t_{m2} - t_{m1}\), the round-trip time.
- **out** as \(t_{s1} - t_{m1}\), the out to slave trip time, and
- **back** as \(t_{m2} - t_{s1}\), the back to master trip time.

For Titan and Edison, Timeline uses the TimeStamp Counter (TSC) register, found in Intel Pentium and later x86 compatible architectures, which counts CPU cycles. The TSC register tracks time at high resolution, for example, 1 nanosecond for a 1-gigahertz processor. Reading the TSC counter is accomplished with a call to `clock_gettime()`. Arguments to this call can specify `CLOCK_MONOTONIC_RAW` which returns the current TSC value scaled to nominal nanoseconds via a linear equation. The slope and intercept used by `clock_gettime()` are updated periodically by NTP. Other argument choices include `CLOCK_MONOTONIC` which returns the current TSC value scaled to track International Atomic Time (TAI — effectively, UTC without adjustments for leap seconds), in nanoseconds; and finally `CLOCK_REALTIME` which returns `CLOCK_MONOTONIC + (now-1/1/1970) + leap seconds`.

Updating this register is a hardware operation, and reading it and storing its value is also fast. Access to the TSC clock is accomplished by calling `MPI_Wtime()`. During Timeline initialization, each node's starting time is collected. During the subsequent experiment’s duration, each node’s time is computed as an offset to that node’s starting time. Each of these new measurements reveals the rate at which the TSC clock is advancing relative to a designated reference node. This results in a clock offset, or \(\theta\). If \(\theta\) equals zero, the trip time \(t_{m2} - t_{m1}\) will be approximately twice that of the out time \(t_{s1} - t_{m1}\). However, if the TSC clock on the remote node has deviated from that of the TSC clock on the reference node, the magnitude of \(\text{out}\), \(t_{s1} - t_{m1}\), will include the amount of the deviation \(\theta\). The value for \(\theta\) is therefore

\[
\theta(t) \approx (t_{s1} - t_{m1}) - \frac{1}{2}(t_{m2} - t_{m1}) = \text{out} - \frac{1}{2}\text{trip}
\]  

(4)

For Mira, a comparable low-overhead clock access is used, the optimized `MPI_Wtime`, and the same reasoning about \(\theta\) applies. Note that the first \(t_{s1}\) occurs immediately after the sync response message in Figure 4. Mira reports \(\text{out}, t_{s1} - t_{m1}\), as a negative value in accordance with the semantics of `MPI_Wtime` (i.e., a globally synchronized time). Indeed, we expect the magnitude of \(\text{out}, t_{s1} - t_{m1}\), to be approximately half the magnitude of \(\text{trip}, t_{m2} - t_{m1}\).

5 | DISCUSSION OF MEASUREMENTS

In the following experiments, the Timeline tool was used to quantify timing characteristics. In all experiments, the master clock is chosen to belong to the rank 0 processing element within the computation, although this choice is arbitrary and any process could be selected to act in this role. For experiments conducted on the Cray supercomputers, Titan and Edison, the locations of all processes on the interconnect topology were recorded. On Titan, the location information is the x, y, z coordinates on the Gemini 3D torus. On Edison, the location information is the coordinates on the Aries dragonfly network. This location information allows postmortem analysis on the effects of relative locations to the clock skew measurements.

Our analysis is divided into two sections. First, in Section 5.1, we make observations regarding interim results that occurred during the course of an experiment. Then, in Section 5.2, we focus on the overall interpretation of our aggregated results.
5.1 Analysis of short time frame results

This subsection presents an analysis of the short time frame results of the experiments conducted on Titan, Edison, and Mira. For this analysis, Table 2 gives an overview of the data used. Per rank (one rank per node), we use 20 measurement replications on Titan, 10 on Mira, and 200 on Edison, each resulting in a similar amount of data because of the different numbers of nodes involved. Measurements are collected with an inner loop over ranks and an outer loop over replications.

Figure 5 depicts the median trip time heatmap for each slave node in the experiment as a function of its physical communication fabric location. Medians are taken over replications. The location of each node, in terms of its coordinates on the communication fabric, was obtained using system library calls. These mapping calls are directly available on the Cray architecture. Accordingly, the figure shows results for Titan and Edison. For both of these cases, coordinates along the XY axes of the communication fabric correspond to horizontal alignment (i.e., parallel to the floor of the data center) while coordinates along the Z axis are vertical. We show the complete supercomputer in both cases to indicate where PBS assigned our runs. Note that a few slots are empty throughout, perhaps indicating blades taken out of service. In terms of physical locations on the floor, the Titan 3-d torus communication fabric is folded and interleaved to control maximum cable length. Figure 5A confirms the torus interconnect with middle nodes in all three dimensions being furthest away from the (0,0) coordinate, showing high latency blue. The results for Edison reflects characteristics of the small diameter dragonfly interconnect which places one subset of slave nodes one hop away from the master and a larger subset of slave nodes two hops away (see Figure 5B).

Figure 6 is the trip time of messages between the master node and each of the slave nodes for each of the slave ranks, showing all replications. The trip time is relative to the master node’s clock only and represents the difference in time stamps between when the master sends the message to the slave and when the master receives the corresponding response from the slave \((t_{m2} - t_{m1})\). This trip time therefore includes both the network latency as well as the internal latency of each slave node but is not contaminated with artifacts related to the offsets between the master node’s clock and each slave node’s clock. The starkly different patterns reflect the different interconnect topology of each architecture. Noting the different scales on the vertical trip time axes, the latencies tend to be the smallest for Edison, followed by Titan, and are the highest for Mira. Variability, however, is the smallest for Mira with most times within 15% of 10 microseconds and the largest for Titan, where they vary by about 50% around 7 microseconds.

Figure 7 shows the out and back times next to the trip times of Figure 6. Note how the vertical scales change drastically for Titan and Edison, squeezing the trip times into a tight line, while changing only modestly for Mira leaving the patterns of trip times still visible. The out and back times include the offset between the slave’s clock and the master’s clock (with opposite signs for out and back, thus the symmetry) in addition to the network latency and the internal latency of each slave node that are also included in trip.

The differences between the three systems in Figure 7 are striking in at least two aspects. The scales on the vertical axes differ by orders of magnitude. The patterns are very different, although some similarities can be interpreted in the two Cray systems.

| System | Nodes used | Replications | Total measurements |
|--------|------------|--------------|--------------------|
| Titan  | 18,000     | 20           | 359,980            |
| Edison | 1,024      | 200          | 204,600            |
| Mira   | 32,768     | 10           | 327,670            |

Figure 5 Heatmaps of median trip times mapped to physical communication fabric locations for Titan and Edison. Coordinate Z increases vertically within each small block.
Edison and especially Titan out and back times completely dominate trip times. Considering Equation 4 for clock offsets, we conclude that Titan's and Edison's out and back times are essentially the clock offsets to about three or four digits of accuracy. The trip term in the equation is negligible when three digits of accuracy are used. This finding is surprising as the clock offsets on Titan are approximately three orders of magnitude larger than the round-trip message latency!

The result for Mira is substantially different as the out and back times do not differ much in magnitude from the trip times. This may be due to the fact that Mira's BlueGene/Q architecture includes hardware-assisted fine-granularity clock synchronization. Our methodology produces an asymmetry on Mira in Figure 7. Using Figure 4, $t_1$ is taken before the message arrives. This skews the distance pair $t_1 - t_0$ (out) and $t_0 - t_2$ (back), approximately by the amount of trip. When the offset is very small on Mira, it results in a negative out and larger and positive back (they are roughly 2.5 and 12.5 with trip being about 10). When the offset is very large (as on Titan and Edison), this is hard to observe. To ensure that the delay on the remote node does not impact our findings, we perform a series of measurements with no intervening activity and utilize the minimum round-trip time from this large set of observations.

To see what patterns exist in the clock offsets, we map the median out times on the physical communication fabric of Titan and Edison in Figure 8. Each recorded out time is already the minimum of ten repeats in a report block (see Sec. 4.2). The median is taken over replications (outer loop) that are separated by polling all other nodes (inner loop). Because of the time span between replications, the offset can drift slightly even in this short time frame analysis so we report the median. Titan clock offsets (Figure 8A) generally increase with distance in the X coordinate. This is unlike the latency heatmap of trip, which reflects the torus interconnect. We suspect that this is an artifact of the NTP synchronization algorithm. A similar conclusion can be reached about Figure 8A for Edison.

### 5.2 Analysis of long time frame results

This subsection is an analysis of the long time frame results of the experiments conducted on Titan and Mira. Here, results for 10 experiments of 10 hours (36,000 seconds) each on Titan using 1024 nodes and one experiment of 7 hours (25,200 seconds) on Mira using 24,000 nodes are considered. For Mira we downsample the data to 600 nodes.
Recall that the out and back times are dominated by node clock offsets on Titan and Edison, as we determined in the short time frame analysis. For this reason, we now refer to graphs of out and back as "offset" graphs. Figure 9 (left) shows the offset $\theta$ for a given node (in this case, rank 1). Each line depicts the offset of rank 1 during a separate 10-hour run (ten runs total). We see that the offset meanders between about 130 and 245 milliseconds with different skews, which occasionally change, probably by the NTP corrections. Each run gives a different pattern, yet the curves are very smooth over the long time scale. The right panel in the figure provides a similar graph for rank 865, chosen arbitrarily.

To get a sense for the offsets of a large collection of ranks in a single run, Figure 10 selects two runs of 1024 nodes to illustrate a typical set and the most extreme set from the 10 experiments. In both Figures 9 and 10, NTP corrections can be seen as discontinuities in the skew.

Because trip, out, and back times on Mira are of roughly the same magnitude, they can be shown in the same graph (Figure 11). What is remarkable on Mira is that the offset is small, several microseconds, and controlled throughout the run duration, especially compared to the Titan offsets.

For some runs, a small subset of the nodes in the computation develops noticeable clock offsets which are eventually mostly resolved by NTP. For other runs, a majority of nodes in the computation develops significant clock offsets. At the worst point, the offset range exceeds 600 milliseconds across some participating nodes in the computation. That is over half a second on a machine that sends messages in about a microsecond! Such a large discrepancy would completely obviate using time-based coordination techniques for things such as fine-granularity operating system scheduling where agreements would need to be on the order of tens of microseconds or less to be effective. In resilience applications, time stamps in system message logs are rendered practically useless for postmortem root cause analysis. Other affected applications would include resolving race conditions in parallel programming environments, optimistic locking strategies in database systems or filesystems, and other similar types of systems programs.

The primary take away from the long time frame results is the ineffectiveness of relying solely on NTP. Similar results were observed for Edison.

6 | RECOMMENDATIONS

Cloud service providers and compute farms have successfully employed multiple strategies to enable distributed time agreement with uncertainties of a few microseconds. Many of the same strategies employed by cloud service providers and distributed computing facilities would seem to
be suitable for deployment on leadership-class machines. Given the demonstrated sensitivity of many parallel workloads and the surprisingly high 0.6-second time uncertainty across a single large-scale parallel machine, we encourage the following best practices.

• First, we recommend replacing NTP-only solutions with higher resolution time synchronization mechanisms. With NTP, accuracies are limited to one millisecond in the best case if a time server is available on the local intranet.30 We believe that improving the resolution of time synchronization within contemporary supercomputers will help to drive system software use cases that leverage tightly synchronized node clocks to improve the performance and efficiency of system software.

• Second, we recommend the use of hardware-based time synchronization mechanisms whenever possible. Hardware-based approaches may be available in high-performance computing environments, for example, the Mellanox Connectx-4 time-stamping.57 For institutions acquiring new machines, we recommend the procurement process as a possible way to get a hardware-based solution.

• Third, in cases where hardware-based time synchronization mechanisms are not possible, perhaps due to cost considerations, we recommend the use of newer software-based time synchronization mechanisms such as RADclock. Software-based approaches are receiving increasing traction in distributed cloud computing environments where point-to-point hardware connections do not exist. Leveraging these approaches in high performance computing environments is likely to be straightforward for a potentially high benefit. One advantage of the time synchronization approach that we advocate is that it can be leveraged against just the nodes and processes participating in a given computation. That is to say, with an approach such as RADclock, the overhead of clock synchronization is always present at the supercomputing system software level. Our approach could be implemented on a per-computation level, thus paying the overhead for tight time synchronization only in instances where it is truly useful.

• Fourth, we recommend that as part of the installation process, the system administrator responsible for installing MPI perform an analysis of available underlying clocks available to MPI_Wtime implementations and document these characteristics for their MPI user community.

• Fifth, we advocate for continuing research that explores new strategies for improving time uncertainty bounds in high-performance computers. As we show that time skew is a smooth function, this provides an opportunity for building predictive statistical and machine learning models that rely on minimal master clock data and give uncertainty bounds. CTP is one method that advances in this direction by arranging nodes that
participate in time synchronization operations in a ring and coordinating time among the ring members. However, within a supercomputer context challenges to approaches such as CTP arise from scalability concerns as the number of nodes in a computation increases.

- Finally, we recommend that customers and vendors in the high-performance computing arena aim for a bounded uncertainty of under 10 microseconds. This recommendation is based on the observation that one-way interconnect latencies in current supercomputers are under 5 microseconds, so achieving clock synchronization of all nodes within 10 microseconds is a readily achievable goal using only software-based synchronization techniques. In cases where hardware-based synchronization techniques are available, even tighter synchronization bounds are likely to be possible.

These recommendations are targeted toward high-performance computers, or systems with the following three characteristics: (a) the machine is intended for applications characterized by large-scale parallelism; (b) the machine contains at least 1000 nodes; (c) the machine contains a communication interconnect with a latency of 5 microseconds or less between any node pair. However, we believe they may prove useful for other distributed and parallel environments as well.

### 7 CONCLUSION

This paper has presented a statistical analysis of the default clock agreement of three representative leadership-class computer systems. Among our findings, we observe that the NTP-based time agreement system allows discrepancies exceeding acceptable uncertainty requirements for many potential applications. In some cases, discrepancies of hundreds of milliseconds are observed, including a worst case measurement exceeding 600 milliseconds across some nodes of a computation. These large discrepancies would prohibit using time-based coordination techniques for things such as fine-granularity operating system scheduling where agreements would need to be on the order of tens of microseconds or less to be effective. Other affected applications would include resolving race conditions in parallel programming environments, optimistic locking strategies in database systems or filesystems, and other similar types of systems programs.

Based on the results presented in this paper, we have provided a set of recommendations that call for the removal of unaltered NTP as the primary tool for time agreement on high performance computers.

To date, our work has assumed symmetric timings for both outbound and return trips of selected round-trip messages traveling between two peers. Specifically, we have assumed those messages chosen from a large set of transfers with minimum round-trip time have symmetric outbound and return timings. In the future, we intend to incorporate methods that characterize asymmetric influences (e.g., adaptive routing, network traffic, OS noise, and so on) as well as incorporate minimum message delay strategies into our investigations.

### ACKNOWLEDGMENTS

The authors would like to thank Don Maxwell of the National Center for Computational Science for his assistance in collecting these numbers. The graphics and statistical analysis contained in this paper were done with the R language and environment for statistical computing.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. In addition, this research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Finally, this research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

### ORCID

Terry Jones http://orcid.org/0000-0003-2187-9707

### REFERENCES

1. Veitch D, Ridoux J, Korada SB. Robust synchronization of absolute and difference clocks over networks. *IEEE/ACM Trans Networking (TON)*. 2009;17(2):417-430.
2. Mills DL. Internet time synchronization: the network time protocol. *Commun IEEE Trans*. 1991;39(10):1482-1493.
3. Top 500 Supercomputing Sites. https://www.top500.org. Accessed: 04-2015.
4. Valiant LG. A bridging model for parallel computation. *Commun ACM*. 1990;33(8):103-111.
5. Oliner AJ, Kulkarni AV, Aiken A. Using correlated surprise to infer shared influence. In: 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), Chicago, IL: IEEE; 2010:191-200.
6. Oeste S, Knüfer A, Ilsche T. Towards parallel performance analysis tools for the openSHMEM standard. *Workshop on OpenSHMEM and Related Technologies*, Annapolis, MD: Springer; 2014:90-104.
7. Marangos N, Rizomiliotis P, Mitrou L. Time synchronization: pivotal element in cloud forensics. *Security and Communication Networks*. 2014;9(6):571-582.
8. Mondragon OH, Bridges PG, Levy S, Ferreira KB, Widener P. Scheduling in-situ analytics in next-generation applications. In: *Cluster, Cloud and Grid Computing (CCGrid)*, 2016 16th IEEE/ACM International Symposium on, Cartagena, Colombia: IEEE; 2016:102-105.
9. Mondragon OH, Bridges PG, Jones T. Quantifying scheduling challenges for exascale system software. In: Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers, Portland, OR: ACM; 2015:1-8.

10. Jones T, Dawson S, Neely R, et al. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In: Supercomputing, 2003 ACM/IEEE Conference, Phoenix, AZ: IEEE; 2003:10-10.

11. Feitelson DG, Rudolph L. Gang scheduling performance benefits for fine-grain synchronization. J Parallel Distrib Comput. 1992;16(4):306-318.

12. Brightwell R, Oldfield R, Maccabe AB, Bernholdt DE. Hobbes: Composition and virtualization as the foundations of an extreme-scale os/r. In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers, Eugene, OR: ACM; 2013:1-8.

13. Akkan H, Lang M, Liebrock L. Understanding and isolating the noise in the linux kernel. Int J High Perform Comput Appl. 2013;27(2):136-146.

14. De P, Kothari R, Mann V. Identifying sources of operating system jitter through fine-grained kernel instrumentation. In: 2007 IEEE International Conference on Cluster Computing, Austin, TX: IEEE; 2007:331-340.

15. Jones T. Linux kernel co-scheduling and bulk synchronous parallelism. Int J High Perform Comput Appl. 2012;26(2):136-145.

16. Ferreira KB, Brightwell R, Bridges PG. Characterizing application sensitivity to OS interference using kernel-level noise injection. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08), Austin, TX: 2008:12 pp.

17. Seeal Lam L, Tantawi A, Lewars J, Divirgilio J, Gildea K. Extreme scale computing: modeling the impact of system noise in multi-core clustered systems. J Parallel Distrib Comput. 2013;73(7):898-910.

18. Mondragon OH, Bridges PG, Levy S, Ferreira KB, Widener P. Understanding performance interference in next-generation hpc systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT: IEEE Press; 2016:384-395.

19. Levy S, Ferreira KB, Widener P, Bridges PG, Mondragon OH. How I learned to stop worrying and love in situ analytics. In: EuroMPI 2016, Edinburgh, United Kingdom; 2016:140-153.

20. Hammouda A, Siegel AR, Siegel SF. Noise-tolerant explicit stencil computations for nonuniform process execution rates. ACM Trans Parallel Comput. 2015;2(1):1-33.

21. Corbett JC, Dean J, Epstein M, et al. Spanner: Google's globally distributed database. ACM Trans Comput Syst (TOCS). 2013;31(3):1-22.

22. Liskov B. Practical uses of synchronized clocks in distributed systems. Distrib Comput. 1993;6(4):211-219.

23. Hoefler T, Schneider T, Lumsdaine A. Characterizing the influence of system noise on large-scale applications by simulation. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, New Orleans, LA: IEEE Computer Society; 2010:1-11.

24. Fidge C. Logical time in distributed computing systems. Computer. 1991;24(8):28-33.

25. Elson J, Girod L, Estrin D. Fine-grained network time synchronization using reference broadcasts. ACM SIGOPS Oper Syst Rev. 2002;36(SI):147-163.

26. Lamport L. Time, clocks, and the ordering of events in a distributed system. ACM Trans Comput Syst (TOCS). 1978;21(7):558-565.

27. Mattern F. Virtual time and global states of distributed systems. ACM Trans Comput Syst (TOCS). 1991;24(8):28-33.

28. DeRose L, Poxon H. A paradigm change: from performance monitoring to performance analysis. In: Computer Architecture and High Performance Interface Users Group Meeting, Dublin, Ireland: Springer; 2008:202-209.

29. Becker D, Linford JC, Rabenseifner R, Wolf F. Replay-based synchronization of timestamps in event traces of massively parallel applications. In: Parallel Processing-Workshops, 2008. ICPP-W508. International Conference on, Portland, OR: IEEE; 2008:212-219.

30. Mills DL. On the accuracy and stability of clocks synchronized by the network time protocol in the internet system. ACM SIGCOMM Comput Commun Rev. 1989;20(1):65-75.

31. Ridoux J, Veitch B, Broomhead T. The case for feed-forward clock synchronization. Networking, IEEE/ACM Trans. 2012;20(1):231-242.

32. RADClock Installation. http://www.synchron.db.org/radclock/installation_linux. Accessed: 02-2017.

33. EEE 1588-2008. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. T echnical Report, IEEE; 24 July 2008:pp. 289. https:/ /doi.org/10.1109/IEEESTD.2008.4579760.

34. Doleschal J, Knüpfer A, Müller MS, Nagel WE. Internal timer synchronization for parallel event tracing. European Parallel Virtual Machine/Message Passing Interface Users Group Meeting, Dublin, Ireland: Springer; 2008:202-209.

35. Jones T, Koenig GA. A clock synchronization strategy for minimizing clock variance at runtime in high-end computing environments. In: Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on, Petrópolis, Rio de Janeiro, Brazil: IEEE; 2010:207-214.

36. Hong C-Y, Lin C-C, Caesar M. Clockscalpel: understanding root causes of internet clock synchronization inaccuracy. Passive and Active Measurement, Atlanta, GA: Springer; 2011:204-213.

37. Maillot E, Tron C. On efficiently implementing global time for performance evaluation on multiprocessor systems. J Parallel Distrib Comput. 1995;23(2):184-213.

38. Gurewitz O, Cidon I, Sidi M. Network classless time protocol based on clock offset optimization. IEEE/ACM Trans Networking (TON). 2006;14(4):876-888.

39. Gurewitz O, Cidon I, Sidi M. One-way delay estimation using network-wide measurements. IEEE/ACM Trans Networking (TON). 2006;14(5):2710-2724.

40. Jeske DR. On maximum-likelihood estimation of clock offset. IEEE Trans Commun. 2005;53(1):53-54.

41. Mondragon OH, Bridges PG, Jones T. The gemini system interconnect. In: High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on, Mountain View, CA: IEEE; 2010:83-87.
47. Faanes G, Bataineh A, Roweth D, et al. Cray cascade: A scalable hpc system based on a dragonfly network. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT: IEEE Computer Society Press; 2012:103.
48. Cray XC Data Sheet. http://www.cray.com/Products/Computing/XC/Specifications-XC30.aspx, Accessed: 03-2015.
49. Alverson B, Froese E, Kaplan L, Roweth D. Cray xc series network. Cray Inc., White Paper WP-Aries01-1112; 2012.
50. Kumaran K. Introduction to Mira. In: Code for Q Workshop, Lemont, IL; 2012:24 pp.
51. Morozov V, Kumaran K, Vishwanath V, Meng J, Papka ME. Early experience on the blue gene/q supercomputing system. In: Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, Boston, MA: IEEE; 2013:1229-1240.
52. Milano J, Lembke P, et al. IBM System Blue Gene Solution: Blue Gene/Q Hardware Overview and Installation Planning, Armonk, NY: IBM Redbooks; 2013.
53. Chiu G, Coteus P, Wisniewski R, Sexton J. BlueGene/Q Overview and Update. https://www.alcf.anl.gov/files/IBM_BGQ_Architecture_0.pdf, Accessed: 04-2015.
54. Kim J, Dally WJ, Scott S, Abts D. Technology-driven, highly-scalable dragonfly topology. In: ACM SIGARCH Computer Architecture News, Vol. 36, New York, NY: IEEE Computer Society; 2008:77-88.
55. Mink A, Carpenter RJ, Courson M. Time synchronized measurements in cluster computing systems. Parity. 2000;1:1-7.
56. Mizrahi T, Moses Y. Serving time in the cloud: Why time-as-a-service? In: Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on, San Francisco, CA: IEEE; 2016:95-96.
57. Configuring Mellanox ConnectX for PTP. https://community.mellanox.com/docs/DOC-2403, Accessed: 02-2017.
58. R Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria; 2015.

How to cite this article: Jones T, Ostrouchov G, Koenig GA, Mondragon OH, Bridges PG. An evaluation of the state of time synchronization on leadership class supercomputers. Concurrency Computat Pract Exper. 2018;30:e4341. https://doi.org/10.1002/cpe.4341

https://doi.org/10.1002/cpe.4341