HARVEY MUDD COLLEGE

Computer Science Clinic

Final Report for
Red Hat

Distributed Point-in-Time Consistent Snapshots

April 7, 2015

Team Members

Philip Davis
Nick Carter
Matt Cook
Michael Saffron (Project Manager)

Advisor

Beth Trushkowsky

Liaisons

Ian Colle
Greg Farnum
Sam Just
Sage Weil

Abstract

Your abstract should be a *brief* summary of the contents of your report. Don't go into excruciating detail here—there's plenty of room for that later.

If possible, limit your abstract to a single paragraph, as your abstract may be used in promotional materials for the Clinic.

Contents

List of Figures

Chapter 1

Introduction

Ceph is a highly distributed, strongly consistent file system. It stores data redundantly within a data center using a type of hashing algorithm. This hashing algorithm determines how data is apportioned to storage nodes. By calculating hash information independently, a Ceph client may communicate directly with the correct storage node without having to consult a separate controller node. The result of this approach is a highly distributed control structure, free of single points of failure, which allows Ceph clusters to be very robust. This communications structure also has the benefit of improving overall performance and scalability by spreading control and communication across all nodes.

For this project, we were tasked with determining a means in which to asynchronously geo-replicate the contents of a Ceph cluster. The task also specifies requirements for performance and consistency. We were able to find a solution to the problem presented to us, and our completed goals are in line with those that we set at the beginning of the project (TODO reference Appendix I). In this document, we are including the theoretical proof that this solution is correct and performant (TODO reference Proof), as well as testing that shows the same thing (TODO reference Analysis). In addition to our promised deliverables, we are also delivering a set of utilities and scripts to replicate and extend our analysis.

Chapter 2

Description of Problem

As Ceph nodes must store data in several replicas, synchronous data replication within a data center is fundamental to the operation of Ceph. Ceph currently does not implement a manner in which to asynchronously replicate a total data set across geographically separated data centers. The design of such an asynchronous replication feature is the goal of this project. This feature would enable usage scenarios such as data center fail-over. One of the largest challenges in implementing this feature is obtaining a consistent snapshot of the total data set at a given point in time. A consistent snapshot here means a partition of the set of events (reads and writes) in the system such there there is no event excluded from the snapshot that an event included in the snapshot depends on. This consistency is necessary from the point of view of a client to the file system.

As Ceph is a distributed file system, without controller nodes that have a full picture of the system, there are inherent challenges to acquiring a complete picture of the system at any given moment in time. Ceph is also strongly consistent, so a potential solution is further complicated as all data that a particular piece of data relies on must also be available. For this problem, a snapshot is allowed to be slightly out of date, but it must present a view of the file system that the file system could have been in at some point in the past. To have older but consistent data is preferable to newer but inconsistent data. This consistency must hold from the perspective of a client. This means that the system must not assume that it has full knowledge of event dependencies, and the system must take into account the possibility of out-of-band communication between clients. For example, there could be multiple users or hosts using the same Ceph distributed file system. These clients could be communicating out-of-band with each other (and, as a result, writing to the file system) without the file systemâĂŹs knowledge. In this case, the file system would not know that these events depend on each other, yet we would still need to ensure that our snapshots are consistent.

The Ceph architecture is designed to grow to hyper-scale. As a result, scalability and performance are of primary concern. The time necessary to snapshot in any proposed solution must not severely increase as more clients and nodes are added. In some production scenarios, snapshots must occur with a minimum frequency regardless of cluster size. Similarly, snapshots must have negligible impact on users. For example, one way to provide a consistent snapshot is to simply block all reads and writes to the Ceph file system while the state of the system is gathered. However, this solution would have a substantial impact on input/output operations, making this solution infeasible.

At the scale of a Ceph deployment, node failures are routine. Any solution must take this into account and treat failures as the rule, not the exception. A solution must consider the impact of node failures on the consistency of a snapshot and be able to adapt accordingly. Similarly, a potential solution must assume that components on which it relies can fail at any time.

Chapter 3

Related Work

Creating a consistent snapshot of a Ceph cluster is difficult because Ceph is inherently distributed. The system does not have a single or small group of controller nodes that track every transaction. Instead, the cluster must aggregate knowledge from thousands of nodes to capture a complete point-in-time snapshot of the data in the cluster. Significant research efforts have been directed toward taking snapshots and synchronizing events in distributed systems. The approaches we reviewed, however, are not by themselves viable solutions to replication problem this project addresses.

3.1 Logical Clocks

Lamport clocks use a logical clock to establish a partial ordering of events in distributed systems. If an ordering of the events and messages between nodes in a distributed system can be established, and this ordering is complete enough to take into account all dependencies, a snapshot may be obtained by observing which events happen before other events. Lamport clocks try to order events and messages by a logical timestamp that increments between each event or message pass, rather than by some physical time that could be affected by clock drift. This ordering method works well if the system can reason about events that are occurring between its own nodes. However, this type of partial ordering fails to guarantee correct ordering in the case of out-of-band communication; the system cannot order the events that occur outside of the system as it has no knowledge of the order of those events. About events that occur strictly within the Ceph cluster, Lamport clocks may still be useful for reasoning about order, but they would have to be used in conjunction with some notion of physical time, or a method of guaranteeing that dependent data from out-of-band communications can still be correctly ordered (TODO Cite Lamport, 1978).

Vector clocks function very similarly to Lamport clocks. However, rather than storing a single logical time, it stores a vector of times that each represent the last time seen on a given node in the system. The ordering of events is based on the relative times stored in the vectors. All messages contain the vector of the process that sent it and a process updates its vectors when it receives a message. The benefit of using vector clocks over Lamport clocks is that vector clocks do not assign an arbitrary time to the event. Instead, it uses the relative times of the processes to determine the partial order of the events. However, like Lamport clocks, out-of-band communication is not accounted for and events could be incorrectly ordered (TODO cite Fidge, 1988).

3.2 Network Time Synchronization

Another approach to event ordering is to use timestamps. This approach has the benefit of a, theoretical, total ordering including in scenarios with out-of-band communication. However, this approach is not as simple as one would first expect. As multiple time sources are in use in a distributed system, the clocks must be synchronized to a high degree of accuracy in order to use timestamps to order events across nodes.

A naive approach to a distributed snapshot algorithm uses timestamps alone to establish a total ordering of events in a system. This works well in systems with centralized controllers or logs that have knowledge of all events. However, in a system with highly distributed control like Ceph, each node only sees a small fraction of events. Each node contains its own clock, and these clocks tend to drift perceptibly over time. Clock skew is generated across a distributed system as clocks drift out of sync. Clocks across a cluster must be synchronized regularly for a timestamp based solution to stay temporally consistent.

To illustrate issues with temporal consistency, consider an example with a cluster containing multiple nodes. Within this cluster, consider event i and event j such that j depends on i. Without some method of ensuring bounded clock drifts, it is impossible to guarantee that a given timestamp at a specific node is correct with reference to the entire system. One node (with perhaps a slightly fast clock) may claim that event i occurred at a later point in time than the event really happened according to some observer. Event j, processed by a node with a slightly slow clock, could then âĂIJoccurâĂİ (be timestamped) before event i. A snapshot algorithm might now believe it is acceptable to include event j and not event i. Such a snapshot would be inconsistent.

A synchronization algorithm could be effective, however, should it provide sufficient, provable bounds on the drift of a given clock and on the skew across the distributed system. These bounds would allow the file system to gain a clear understanding of what knowledge of event ordering it has when taking a snapshot.

Many clock synchronization algorithms already exist and are in wide use. However, these algorithms tend to have major shortcomings when they are considered for application to this problem in a hyper-scale data center. A synchronization method with provable bounds and reasonable communication complexity is required if we are to use only timestamps for ordering events in a snapshot. We must also be able to prove the correctness of the error bound calculations in order for Ceph to make guarantees about the consistency of its snapshots.

3.2.1 Network Time Protocol

NTP is a robust algorithm for time synchronization (TODO D. Mills et al., 2010). Currently it is among the most popular time synchronization algorithms. NTPâĂŹs is designed synchronize geographically disparate computers over the internet. As a result, it is resilient to node failure, network inconsistencies, and poor clock quality. NTP requires very little information about the network and nodes on which it is operating in order to provide useful synchronization. It uses a number of statistical estimators to predict future clock performance from previous performance. Of particular note to our algorithm, NTP does define a maximum error term in relation to a single, root time source. As a result, NTP is an appropriate choice for a network time protocol in our algorithm.

A better solution is possible, however. NTPâĂŹs focus on robustness causes compromises for the freeze time. NTP consistently overestimates uncertainty. A protocol more explicitly designed for local area network synchronization could likely do away with some of the complexity of NTP, make more assumptions about network configuration, and as a result provide tighter bounds on the time.

In this report, we analyze the performance of an NTP implementation, ntpd, across various clock and network conditions. The ntpd daemon was chosen for it common use and ease of access to relevant calculated parameters <TODO citation>. Chrony is another potential choice for an NTP implementation <TODO citation>, although it lacks some diagnostic reporting.

3.2.2 Precision Time Protocol

PTP claims to be good for tightly synchronizing computers on a local network <TODO citation>. Our preliminary testing suggested that this was the case. To gain even better synchronization, a user may use specialized hardware (that is relatively commonly supported and currently in use) to decrease the amount of random variation in message latency in a network. This allows the protocol to get extremely accurate measurements of time, and in theory also extremely tight uncertainty bounds. However, the protocol does not specify, and the implementation does not include, an upper bound error value. This means that this protocol is not currently suited for use with our algorithm.

If performance of NTP is found to be unsatisfactory however, it would be worth considering extending PTP (<TODO reference implementation details>) – the implementation of a maximum error term – would be advisable. Linux PTP is an implementation of the Precision Time Protocol (PTP) <TODO citation>. PTPd is another implementation <TODO citation>.

3.2.3 Wireless Synchronization

Surprisingly, wireless time synchronization is a much easier problem. Wireless signal propagation times are very easy to model and as a result time synchronization is easy to perform with a very high level of accuracy. Protocols such as PulseSync take advantage of this observation <TODO citation>.

If extremely small freeze windows are a requirement, a wireless synchronization protocol could easily be designed that would allow for a very high level of confidence. However, this would necessitate a significant amount of specialized hardware. This would be counter to the projectâĂŹs goal of getting the most performance possible out of commodity hardware.

GPS Synchronization is a special case of wireless synchronization that can provide a nearly perfect time source anywhere around the world. It is advisable to incorporate a GPS time source (or similarly accurate time source) into the master clock of any implementation of our algorithm <TODO REFERENCE IMPLEMENTATION SECTION>.

3.3 TrueTime

Google has a number of very time-sensitive applications, most notably the synchronously georeplicated Spanner database. Synchronous georeplication of database transactions requires very strong time guarantees because database transaction ordering is important. A library called TrueTime was developed to support these applications.

3.3.1 TrueTime.now()

TrueTime does not provide current time in the way that most time libraries do. Instead, it gives a range that is guaranteed to contain the current time. The bounds are claimed to be generally less than 10 milliseconds. This allows for very rapid throughput on the database while still maintaining definitive transaction ordering. When TrueTimeâĂŹs guaranteed bounds start to spread, it will throttle writes to maintain that ordering (TODO Corbett et al., 2012).

3.3.2 TrueTime's Clocks

TrueTime relies on having extremely good clocks to maintain sub-10ms skew between geographically diverse data centers. Specifically, Google has placed atomic and GPS clocks in their data centers. TrueTime runs a daemon that talks to these clocks, both in its own data center and in others. A variation on MarzulloâĂŹs algorithm is used to weed out clocks whose timing information is not reliable (e.g. due to network latency) (TODO Corbett et al., 2012).

3.3.3 Applicability

As a concept, TrueTime is interesting. However, as it is a proprietary library and as it requires special hardware, it would not be able to be used directly as a solution to CephâĂŹs asynchronous replication problem. However, if a similar open source protocol became available, it could merit deeper analysis.

Chapter 4

Approach

We have developed a simple, performant clock-synchronization-based algorithm that can provide consistent data center level snapshots.

The algorithm has a few requirements. First, it requires that the clock synchronization algorithm used by the data center support error bounding. The most common NTP implementation, ntpd, supports this. Second, to remain performant it requires a certain baseline clock and network link quality, although it will degrade gracefully with poor clocks or links (TODO reference results section)

The algorithm we propose has 4 phases. They are as follows:

1. *Synchronization*

   This âĂIJphaseâĂİ is happening continuously in the data center. All nodes sync their clocks to a common master clock using a synchronization algorithm supporting error bounds as discussed above. Additionally, snapshot times are pushed out (possibly via map updates or embedding in heartbeat replies).

2. *Freeze*

   Nodes hold incoming writes. The writes may be processed, but completion is not acknowledged. Let U_i be the uncertainty in node *i*âĂŹs current time, and T be the scheduled snapshot time. Node i begins its freeze when its clock reads $T - U_i$, and completes it when its clock reads $T + U_i$, guaranteeing that the master clockâĂŹs T is captured in the freeze window for all i. (TODO reference proof section)

3. *Confirmation*

Before a snapshot may be marked as good, the data center must wait a short period. This allows any sudden clock desynchronization events or node failures to be detected, and consistency checked. Specifically, RADOS verifies that for each failure, at least one other member of all the PGs on that node did not fail or have clock desynchronization events. If no errors are found, the snapshot is marked as good

4. *Replication*

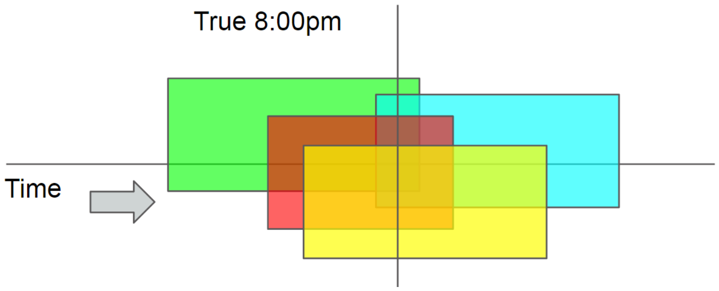
   Finally, if the snapshot was marked as good after phase 3, the data may be replicated. Because the first step in taking a snapshot is simply writing down a marker in an OSDâĂŹs log, it is straightforward either to do a full snapshot, or just a diff since the last good snapshot.

4.1 Proof

4.2 Overlapping Freeze Windows

The freeze windows for our algorithm are chosen specifically to ensure that there is some point in time when all of the nodes are frozen. Each node knows the intended freeze time. However, since their clocks arenâĂŹt perfect, they donâĂŹt know exactly when it is.

Recall that our algorithm calls for each node to start its freeze window when its clock reads $T - U_i$, and completes it when its clock reads $T + U_i$, where T is the freeze time and U_i is NTPâĂŹs uncertainty at that time. Since NTP guarantees that the real time T will be within the uncertainty bounds of when the nodeâĂŹs clock reads T, we can be sure that each node will be frozen at the freeze time. Therefore, we can be certain that their freeze windows overlap. The figure below demonstrates four freeze windows at the freeze time of 8pm.



4.3 Consistency

Our algorithm is designed to take consistent snapshots. A snapshot would be inconsistent if it captured an event, but not an earlier event that caused it. Let A be some event that caused B, another event. For example, A could be a new class that weâĂŹre adding to a library and B could be unit tests for that class. If we captured B in our snapshot, but not A, our snapshot would contain the unit tests for a class that doesnâĂŹt exist. This would cause the test suite to fail or crash, which could be very bad. Therefore, itâĂŹs important that, if our snapshot captures an event, that it also captures all events that caused it as well. Formally, here is the statement that we want to prove:

Let A and B be events in our system, where B was caused by A. If B was captured in our snapshot, then A must have been as well.

Firstly, letâĂŹs define T_A to be the time at which A occurred and T_B to be the time at which B occurred. Since A caused B, we know that A must have taken place before B, meaning $T_A < T_B$.

Our solution guarantees that all of the freeze windows of the nodes in our system will overlap at some point in time, meaning that, for some point in time, all nodes will be frozen. Now, itâĂŹs likely that the freeze windows will all overlap for some interval, not for only a single moment in time. During this interval, the cluster will not respond to write events. Therefore, when we are considering the order of events, we can collapse that whole interval into a single event and we can consider the start time of the interval to be the time of the event. LetâĂŹs call the event F and its time T_F.

LetâĂŹs assume, for the sake of contradiction, that there is a series of events that could violate our proposed invariant. More specifically, letâĂŹs assume that it is possible for B to have been captured in the snapshot while A was not.

Each nodeâĂŹs portion of the snapshot is taken when that node freezes. No other events will occur on the node between when it takes its snapshot (i.e. the beginning of its freeze interval) and F, the moment when all nodes are frozen. Therefore, if B was included in the snapshot, it must have happened before F, meaning $T_B < T_F$. Since A was not captured in the snapshot, it must have happened after F, meaning that $T_F < T_A$. We can combine these two inequalities together to get $T_B < T_A$.

This contradicts the original requirement that A must have taken place before B, or $T_A < T_B$. Therefore, since we have reached a contradiction, we know that our original assumption must have been incorrect and, therefore, our statement must be true.

Since our statement is true for any two events in our system with a causal relationship, we know that our snapshot must be consistent.

4.4 Performance

There are a number of different performance characteristics that need to be considered to ensure that our algorithm will not have an adverse impact on the current functionality of Ceph. First, we need to consider the impact of delaying writes for a freeze. Since each node freezes for the uncertainty that is determined by the time protocol, the magnitude of the uncertainty is dependent on which time synchronization protocol is used. We analyze the uncertainty that NTP provides later in this document and discuss what impact our solution would have on a Ceph instance if NTP is used as the time synchronization protocol. However, if a time synchronization protocol could provide uncertainties in 10-20 millisecond range, giving 20-40 millisecond freeze windows, our algorithm shouldnâĂŹt impact the clusterâĂŹs performance significantly. Also, freeze windows of each of the nodes is offset from each other, which means that the whole Ceph instance is frozen for a much shorter period of time.

Next, we consider the networking requirements of this algorithm. We need to be assured that our algorithm does not require significant amounts of bandwidth which would cause network congestion within a Ceph instance. Since most Ceph instances are already running a time synchronization protocol like NTP, our algorithm does not require any extra messages during the sync/update phase. The freeze phase also does not require any extra messages. The number of success/failure messages required in the confirmation phase is $O(n)$ in the size of the instance and thus would not have a significant impact on the performance of the instance (because each message would be only a few bytes). In the replication phase, the remote instance needs to read the data from the local one, but this cost is inherent to any snapshotting algorithm and thus not a negative characteristic of ours in particular.

Finally, we need to consider whether this algorithm will place a significant computational burden on nodes. Each primary OSD needs to send its PGs' information to the remote instance. This should not impact the performance of the primaries, since it shouldnâĂŹt take too long to send all of that information. Even if it did take a while to send the information, read and write requests could still be addressed by prioritizing those requests above the requests for the objects from the remote instance. Sending all of the information out at once could use a lot of bandwidth, but since it is distributed throughout the Ceph instance, the choke point would be the bandwidth from the local instance to the remote instance, which should be pretty large and not a concern if we prioritize normal client traffic over the inter-instance traffic.

If the user wants to store the uncertainty information of the nodes in the monitors, the monitor nodes will have to store the uncertainty of each node in the local instance. Though there could be tens of thousands of OSDs, storing even eight bytes of time information would only be a couple of kilobytes of data. Even if it took a megabyte to store all of the timestamps, it would not be a significant memory burden. Also, reporting clock information to the monitors would require a constant number of messages for each node and, therefore, only a linear number of messages in the number of nodes in the instance. Based on our analysis, our algorithm should not have a significant impact on the performance of the Ceph instance.

Chapter 5

Data Collection and Results

With a theoretical proof of the core of our algorithm complete, we can move on to an analysis of its real-world implementability and performance. We can first look at whether the tools we use to simulate time synchronization in the real world continue to conform to the invariant set out in our proof. We also can look at the performance of our freezing algorithm and analyze how big of a freeze window is necessary in a real world setting.

5.1 Collection of NTP Performance in a Real-World Setting

There are two primary characteristics of our algorithm that need testing: the existence of the freeze period overlaps described in our solution, and the worst and average case duration of these freeze periods. As described above, we used simulations to observe the existence of the overlap. In order to accurately simulate the characteristics of a real Ceph cluster, we need to model the actual behavior of a given Ceph cluster.

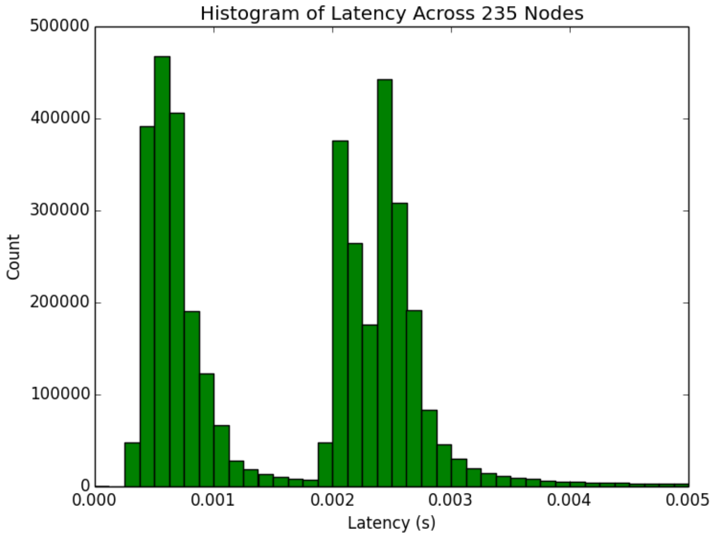
To do so, we used the Network Time Protocol daemon, ntpd, which is an operating system program that maintains the system time in synchronization with the Network Time Protocol (NTP) servers. This allowed us to log data and statistics about the clock and network communications. In particular, the data that interests us for simulation is the clock jitter and network latency.

The clock jitter is the estimated random error on a given computerâĂŹs clock drift from the NTP server, and is typically measured in Parts Per Million (PPM). For example, a PPM of 10 for clock jitter would mean for every million clock ticks of the NTP server, the local computer could be off by 10 ticks. The network latency is a measure of the round trip time of the message passed between the local computer to the NTP server and back. This is typically on the order of milliseconds. <TODO citation> <development>

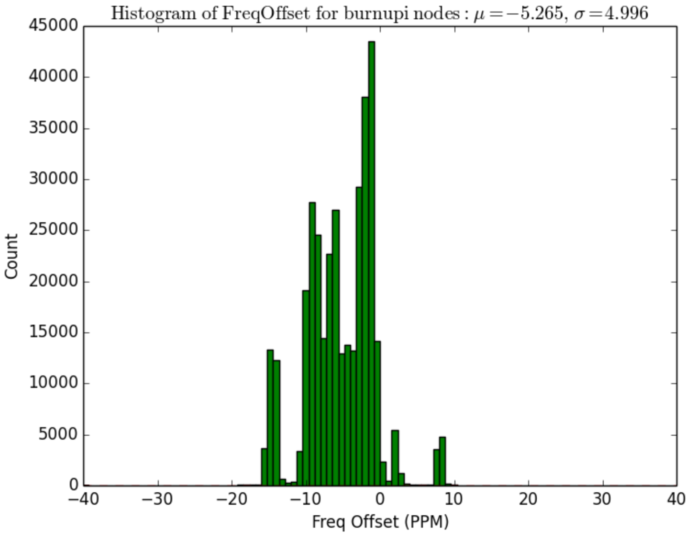
The Ceph test cluster includes three types of computers named Mira, Plana, and Burnupi. Miras and Planas use Intel-based chips, while Burnupi uses AMD chips.

The histogram on figure 5.1 shows the collected values on network latency. Note that there are three peaks that can be seen, and this is reflective of the three different types of computers. Among all three types of computers, network latency in the test cluster appear to generally be around single digit milliseconds.

Figure 5.1 Histogram of network latency across 235 nodes in a Ceph test cluster.



We observed the behavior of clock jitter on the same Ceph test cluster, as shown in Figure A, B, and C for the different types of computers. Despite the different offsets, these can be seen to have a generally normal distribution. These values are reasonable as we would generally expect clock jitter to be at most around ±20 PPM. The average standard deviation for individual Burnupi nodes is 1.061 PPM, Mira nodes is 0.648, and Plana nodes is 2.011. The average across all is 1.237 PPM.

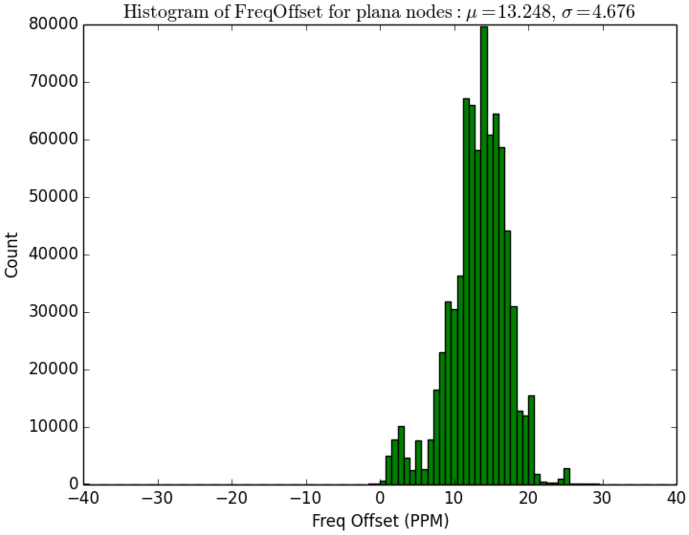


5.2 Simulation of NTP for Performance and Analysis

Although the recorded real-world statistics have been very useful for understanding the behaviors of clocks and for validating our understanding of time synchronization, real-world records are not sufficient to be able to determine if our algorithm is correct. We must also look at simulation results to be able to check that the invariants that we would like to hold true do actually hold true. A simulated network environment running on a single computer has the advantage of having a single clock on which all simulated clocks are based. By having a single clock, we are able to quantity and verify the claims that time synchronization algorithms make about the quality of a synchronization and the quality of the clocks involved.

By using a simulation that abstracts the clock from a physical clock, we also eliminate any potential for idiosyncrasies of the host system clock to affect the measurements gathered in the simulation. This has the added benefit of allowing the simulation to run at a speed much greater than real time.

We have chosen to use an environment called clknetsim <TODO citation> to conduct our simulations. This choice was made for a variety of reasons. Clknetsim is an open-source simulation package developed and used by a major contributor to the chrony project <TODO citation> and the linuxptp project <TODO citation> to test these protocol implementations.



In comparison to commercial network simulation products, It provides us with a simple codebase and featureset targeted at our use case of measuring time across a simulated network.

Clknetsim makes use of LDPRELOAD to intercept system calls that time sychronization libraries make on their hosts. Using these intercepted calls, clknetsim is able to monitor and capture the internal state of the synchronization program. It is also able to fully control the information the time library receives from the network and the system clock. As clknetsim is in simulating system clocks and the network connections, it can shape behavior of these components to test different setups and hardware properties.

We are most interested in collecting and measuring two parameters in this system. We first want to determine if the maximum uncertainty reported by NTP is accurate. Clknetsim allows us to do this by, for a given timestamp in the simulation, allowing us to observe what NTP thinks the time is along with information about how certain NTP is about that time. Our second goal is to analyze how low of an uncertainty can be reached. As our algorithm is predicated on a upper bound of the uncertainty in time, performance analysis can be accomplished by observing how modifying various parameters affect the maximum error term that NTP reports.

Over the course of our project, we have made slight additions to and modifications of the clknetsim package. These modifications and extensions are mostly superficial and made in order to expose state parameters of NTP that were not previously being exposed, including the maximum error term.

Chapter 6

Analysis

6.1 Correctness

When we run our simulations, we want to determine whether NTP can give accurate bounds on each nodeâĂŹs uncertainty range to include the real time. This is the key characteristic of NTP that allows our algorithm to work correctly.

At each node, NTP has an estimate for the real time and an uncertainty in that estimate. Clknetsim allows us to access that information, and it allows us to know the real time of the system. All of this allows us to know what the âĂIJsafety bufferâĂİ is for each node in real time.

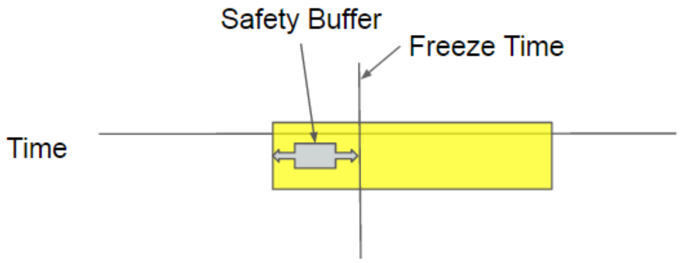
The safety buffer, $S(t)$, for a particular node is defined as:

$$S(t) = U(t) - |t - E(t)|$$

where t is the real time, $U(t)$ is NTPâĂŹs uncertainty in the real time estimate for that node, and E(t) is NTPâĂŹs estimate for what the real time is. In simpler terms, the safety buffer tells us where NTPâĂŹs error in what the real time is lies within its uncertainty range (see figure 6.1. If the safety buffer is a positive value, then the difference between NTPâĂŹs estimate and real time is within its uncertainty range. If it is negative, then itâĂŹs error is outside of its uncertainty range, meaning that NTP doesnâĂŹt function as we would expect.

To prove the correctness of NTP, and therefore our algorithm, we ran a Clknetsim simulation with 10 nodes over xxxxx section seconds, recording the safety buffer for each of the nodes at every moment in time. We then aggregated those values for all of the nodes together and we computed five points of data: the minimum value, the maximum value, the average value,

Figure 6.1



the average value plus one standard deviation and the average value minus one standard deviation. Those values are shown in figure 6.2

We can see that all of these values are positive. Since the minimum value is a positive number, we can be sure that all of the values are positive, which shows that NTP is functioning as we would hope. Therefore, our algorithm can use NTP as a time synchronization protocol.

6.2 Performance

Our testing has shown that NTP is a capable and resilient algorithm. It seems likely that in any real-world network configuration (baring hardware failure) NTP will be able to report accuracy statistics that will result in the correct performance of our algorithm. However, more exacting network layouts will be necessary to make our algorithm performant. The data displayed in this section summarizes trends seen when varying a number of parameters in a standard network setup. We find few surprises – network latency is the standout determining factor for how performant our algorithm will be. Lower the mean network latency and you will be rewarded by a fairly consistently linear drop in freeze times.

ItâĂŹs worth discussing how the network latency is modeled for these simulations. Network latency was modeled using a gamma distribution, since the data from the test Ceph cluster seemed to suggest that latency approximately followed such a model. We varied two parameters: the mean value and the shaping parameter, or alpha.

The shaping parameter represents how much of a tail our gamma distribution has: a larger shaping parameter means that our gamma distribution would have less of a tail. In the extremes, a small shaping parameter approximates an exponential distribution and a large shaping parameter approximates a normal distribution.

The NTP Max Error term is strongly affected by the mean network latency. There is a much weaker effect on the Max Error by the shape of the distribution of network latency traffic. NTP cares very little about how the network tragic is distributed as long as the mean is low. We can see in the upper right of MaxError vs Latency Mean vs Alpha that there is a slight increase in Max Error for more distributed latencies, especially as the mean latency grows.

Interestingly, we see that the mean of the max error term increases with increasing alpha. This is likely because as alpha increases the data become more normal. When the network latency distribution has a long tail, most of our values will be less than or below the mean value. Values larger and farther away from the mean tend to happen less frequently and are more anomalous. Likely, NTP sees these values are anomalies and ignores them, keeping its uncertainty at a lower, more reasonable value. This explains why, for the graphs of the mean and maximum of the error, we see lower values for lower alpha values.

As can be seen in figure 6.3, figure 6.4, and figure 6.5 there is very little, if any, effect of the drift variance of a nodeâĂŹs internal clock on the max error term. This is expected network latency is a much larger term in comparison to even the worst clocks. We again do see a strong, apparently linear, correlation between mean latency and the max error term. Looking at the mean of the max error term, we see values of between 15 and 45 ms approximately. The max of the max error term for any given run appears to be fairly predictable with few outliers in the 20 to 70 ms range. The standard deviation of the measured max error values is consistent with this, showing generally low variability in the spread of max error values. We notice that the spread does increase with increasing latency mean.

Larger latency values do seem to mean that at any given time the absolute time offset may be larger. However, network latency does not seem to play a large role in the mean time offset. Drift seems to play almost no role.

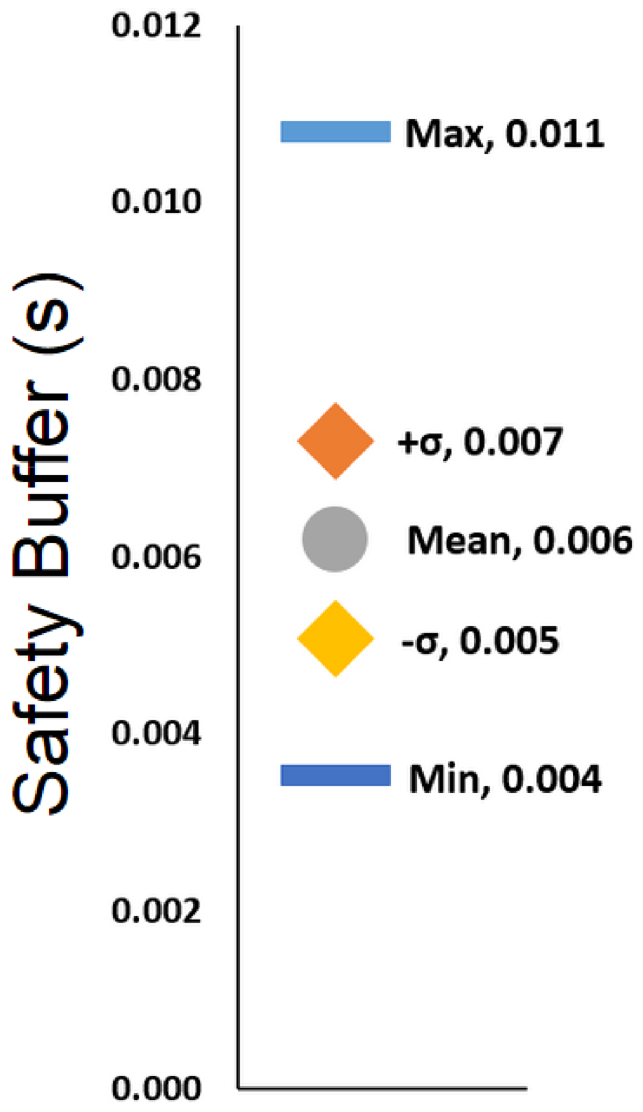
Larger latency values do seem to mean that at any given time the absolute time offset may be larger. However, network latency does not seem to play a large role in the mean time offset. Drift seems to play almost no role.

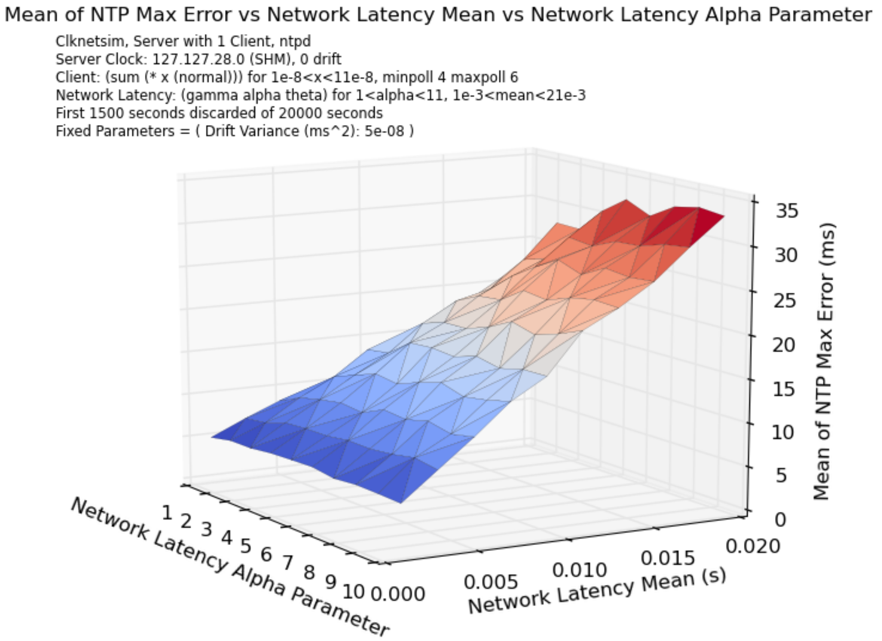
For larger variance in the drift, NTP does have to make larger frequency corrections, but it seems that NTP is more than capable of doing this as these clock discipline corrections do not seem to translate into larger error or larger absolute time offsets. As the mean latency increases, NTP also seems to make larger frequency corrections. This likely means that, due to network asymmetries, NTP is more likely to over correct or undercorrect at times, resulting in an increase in the maximum offset. We also see that more normal latency distributions also result is smaller deviations in the frequency offset. The mean seems to not be significantly affected by either the latency or the drift variance, suggesting that most of the the clock frequency changes are the result of overcorrection and not the result of the clock actually keeping poor time.

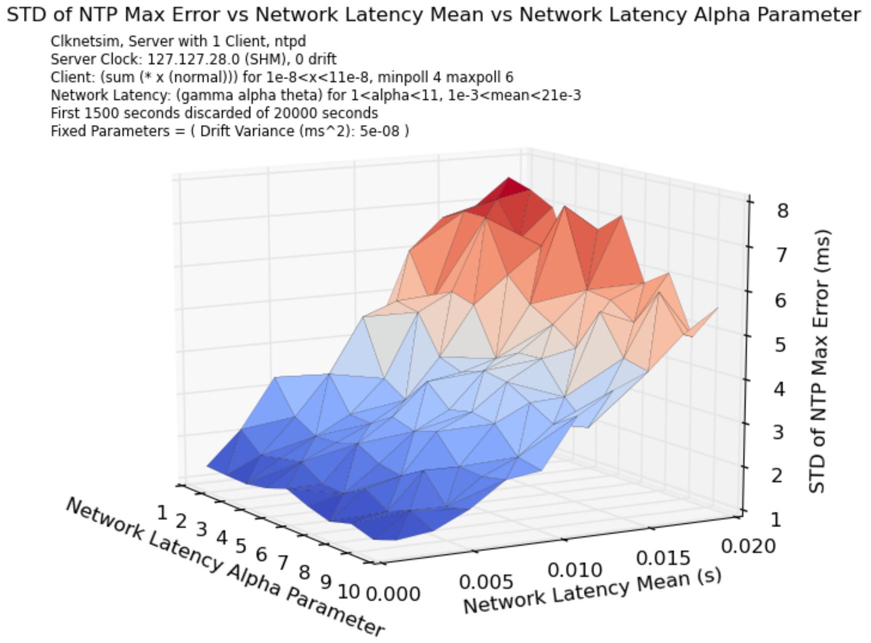
In summary, network latency mean has a larger effect on freeze times. Network traffic shape does have an effect on freeze times, but its effect is small in comparison to mean latency. An individual nodeâĂŹs clockâĂŹs drift variance is inconsequential to the final performance of our algorithm.

Due to how uncertainties compound in NTP, it is highly advisable to acquire a single, good clock to serve as a master. This will significantly decrease freeze windows.

Figure 6.2


Safety Buffer (s)
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Max, 0.011
+σ, 0.007
Mean, 0.006
-σ, 0.005
Min, 0.004


Mean of NTP Max Error vs Network Latency Mean vs Network Latency Alpha Parameter
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Drift Variance (ms^2): 5e-08)
Mean of NTP Max Error (ms)
0
5
10
15
20
25
30
35
Network Latency Alpha Parameter
1
2
3
4
5
6
7
8
9
10
Network Latency Mean (s)
0.000
0.005
0.010
0.015
0.020



STD of NTP Max Error vs Network Latency Mean vs Network Latency Alpha Parameter

Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Drift Variance (ms^2): 5e-08)

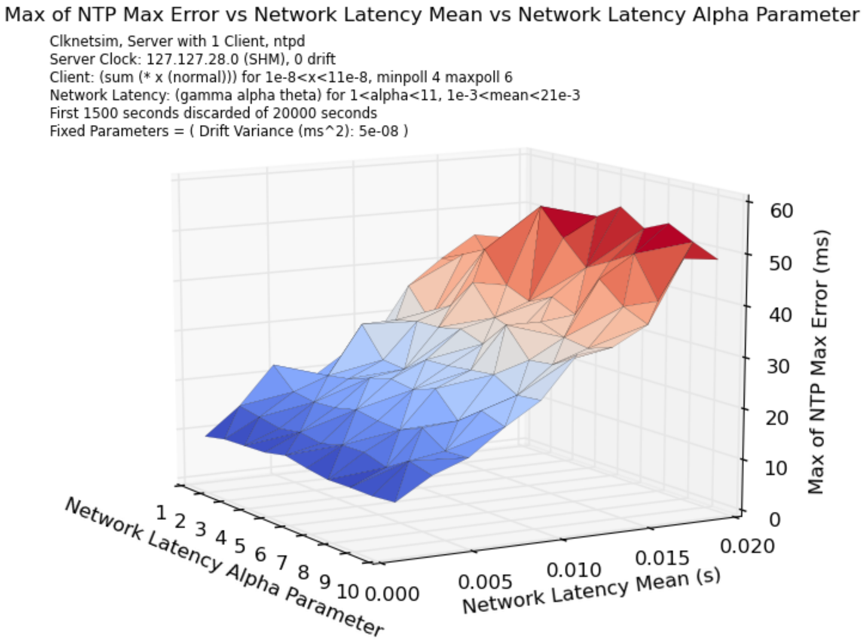

Max of NTP Max Error vs Network Latency Mean vs Network Latency Alpha Parameter
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Drift Variance (ms^2): 5e-08)
Network Latency Alpha Parameter
1
2
3
4
5
6
7
8
9
10
Network Latency Mean (s)
0.000
0.005
0.010
0.015
0.020
Max of NTP Max Error (ms)
0
10
20
30
40
50
60

Figure 6.3

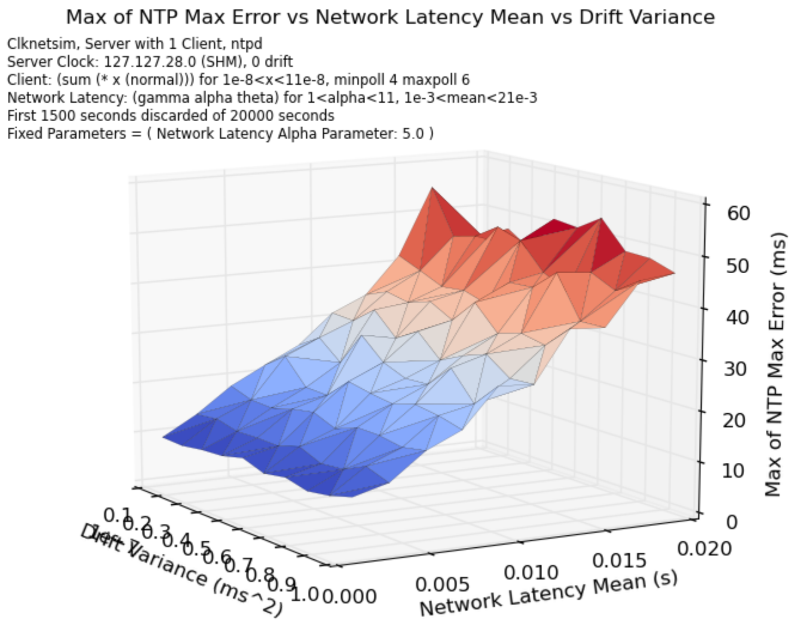

Max of NTP Max Error vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
Max of NTP Max Error (ms)
0
10
20
30
40
50
60
Drift Variance (ms^2)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Network Latency Mean (s)
0.000
0.005
0.010
0.015
0.020

Figure 6.4

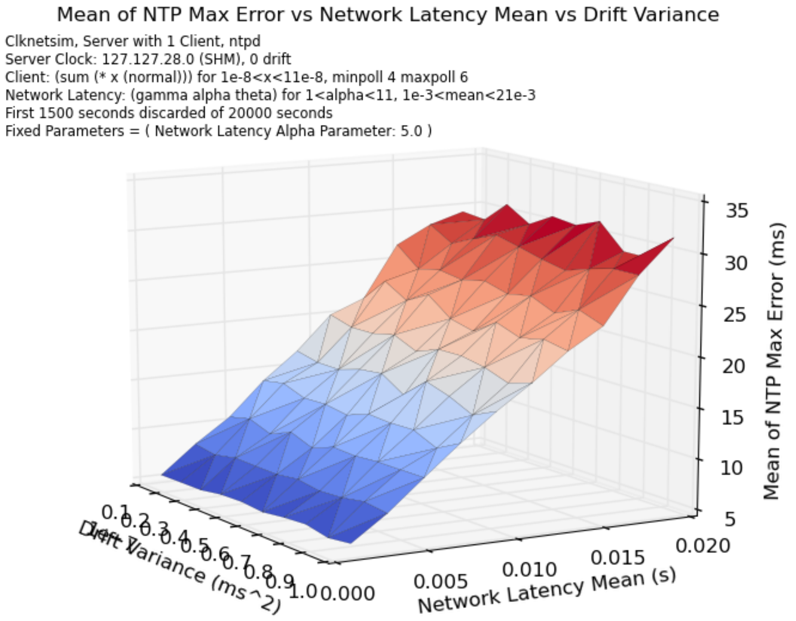
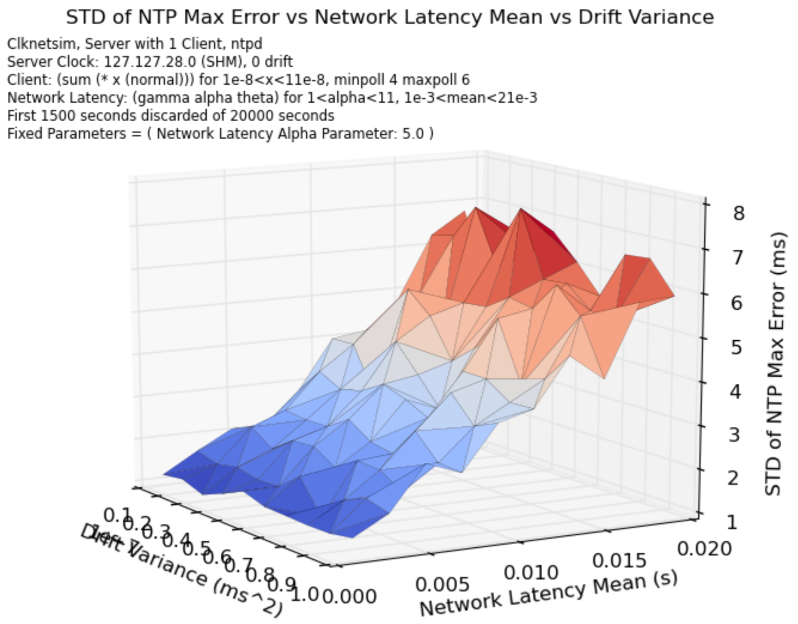
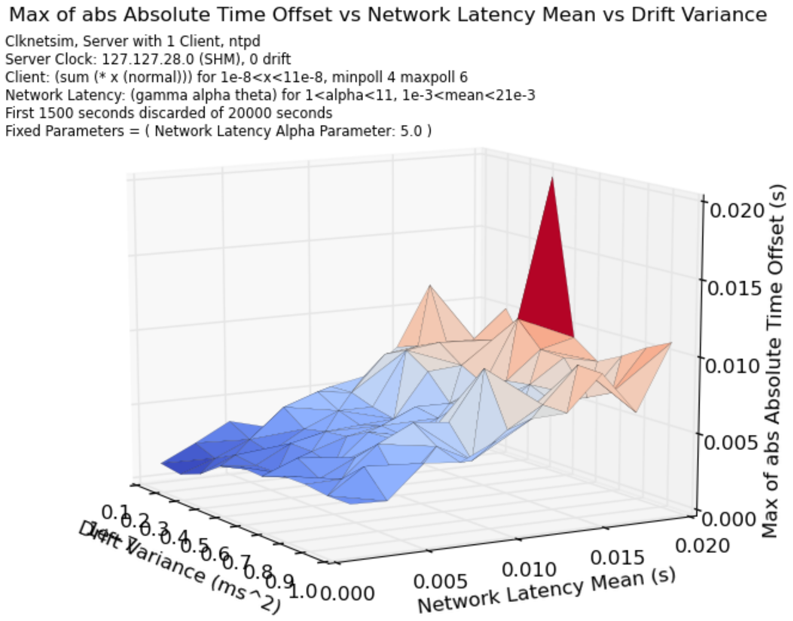

Mean of NTP Max Error vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Drift Variance (ms^2)
0.000
0.005
0.010
0.015
0.020
Network Latency Mean (s)
5
10
15
20
25
30
35
Mean of NTP Max Error (ms)

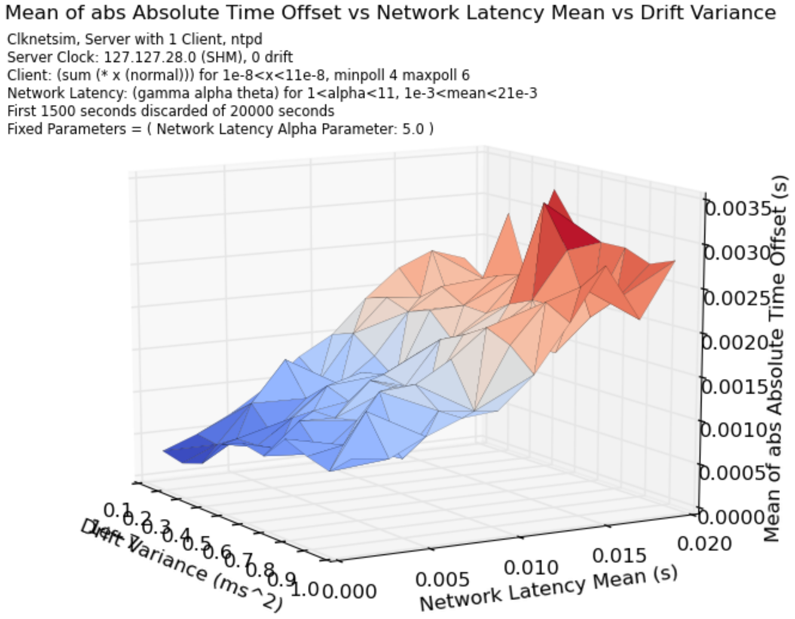
Figure 6.5

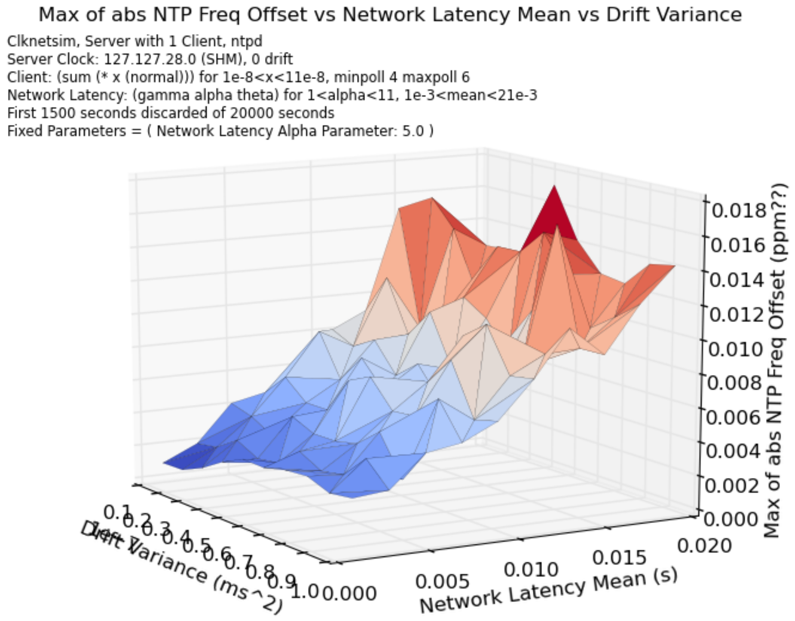


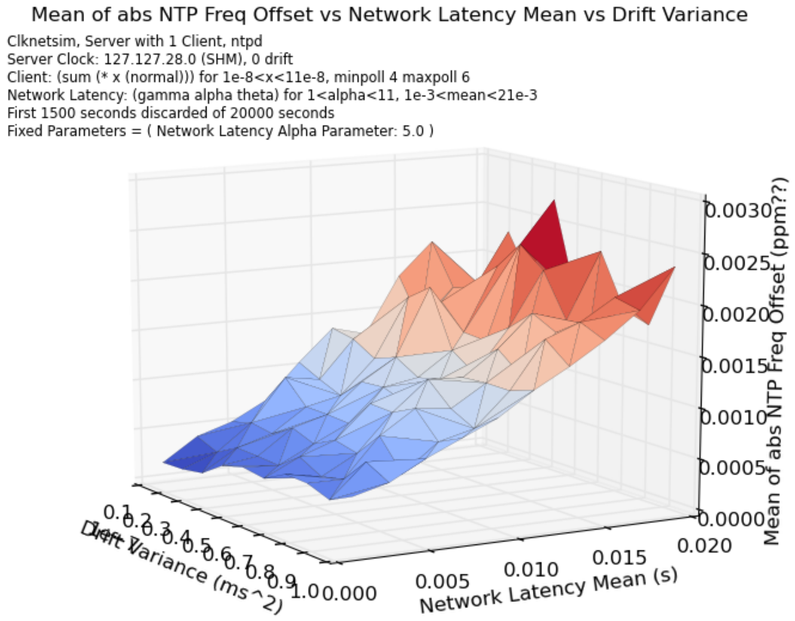
STD of NTP Max Error vs Network Latency Mean vs Drift Variance

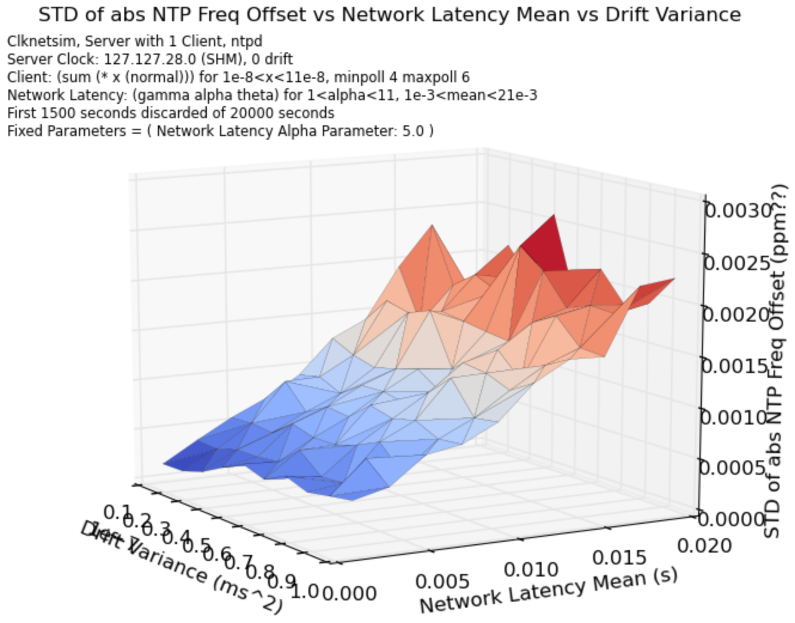
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)


Max of abs Absolute Time Offset vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
Max of abs Absolute Time Offset (s)
0.000
0.005
0.010
0.015
0.020
Network Latency Mean (s)
0.000
0.005
0.010
0.015
0.020
Drift Variance (ms^2)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Mean of abs Absolute Time Offset vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
Mean of abs Absolute Time Offset (s)
0.000
0.005
0.010
0.015
0.020
Network Latency Mean (s)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Drift Variance (ms^2)


Max of abs NTP Freq Offset vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
Max of abs NTP Freq Offset (ppm??)
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Network Latency Mean (s)
0.000
0.005
0.010
0.015
0.020
Drift Variance (ms^2)
1e-7
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


Mean of abs NTP Freq Offset vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Mean of abs NTP Freq Offset (ppm??)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Drift Variance (ms^2)
0.000
0.005
0.010
0.015
0.020
Network Latency Mean (s)


STD of abs NTP Freq Offset vs Network Latency Mean vs Drift Variance
Clknetsim, Server with 1 Client, ntpd
Server Clock: 127.127.28.0 (SHM), 0 drift
Client: (sum (* x (normal))) for 1e-8<x<11e-8, minpoll 4 maxpoll 6
Network Latency: (gamma alpha theta) for 1<alpha<11, 1e-3<mean<21e-3
First 1500 seconds discarded of 20000 seconds
Fixed Parameters = (Network Latency Alpha Parameter: 5.0)
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Drift Variance (ms^2)
0.000
0.005
0.010
0.015
0.020
Network Latency Mean (s)
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
STD of abs NTP Freq Offset (ppm??)

Chapter 7

Implementation

There are a few things to note about how implementation would work in practice. First, NTP requires a few thousand seconds to settle, because it not only works with last-seen error but also estimated drift rate, changes in that drift rate. Thus, a new machine that is being added to the cluster could not be used in snapshots until roughly a few thousand seconds after it was added to the cluster. Probably this would be handled by giving new machines a âĂIJwarm upâĂİ period before they are actually assigned data. Once the machineâĂŹs NTP estimates settle, it can be incorporated into the Ceph cluster.

The quality of the master NTP clock significantly affects the size required for safe freeze windows, as we saw in the simulation results. We recommend that a GPS, atomic, or other highly precise clock (any device capable as acting as an NTP stratum 0) be used instead of the on-board RTC for that machine. This allows the master clock to function as a stratum 1 clock, eliminating NTP uncertainty due to upstream variation. This is not strictly necessary, and performance is generally acceptable with a more distant stratum, but a stratum 1 clock will provide the best performance for the cluster. Without a stratum 0 device directly connect to the NTP stratum 1 server, NTP is forced to assume to worst possible clock properties for the on-board RTC.

Another time protocol could be used in place of NTP, and ideally one more tailored to data center precision would in fact be usable. NTP was used for analysis here because it is widely deployed, it can bound its own uncertainty, and the reference implementation of the protocol provides easy access to important diagnostic information. It is most likely possible to implement the required uncertainty calculation in PTP, but this would require extensions beyond the specification of the protocol. Chrony, another NTP implementation, could likely also be modified to report the necessary information.

There are a few fault conditions that can occur during a snapshot. The simplest, the primary NTP master clock going down, would require a short suspension of snapshotting (a few hundred to a few thousand seconds, depending on the specific characteristics of the clocks) until all clocks in the data center re-settled on the new master clock. Scenarios with more than one master clock on a network have not been analyzed.

7.1 Snapshot Validation

As in all systems, hardware failures are possible. As a result, we must consider how to mitigate against a network disruption or clock hardware failure leading to a significant clock desynchronization of an individual or group of nodes. This would be detectable via a major spike in various diagnostic outputs from NTP (different kinds of uncertainty and measures of clock drift and wander. The node would report this event either within a heartbeat message or through an extra, priority message to one of the monitors. The monitors would then be able to check that all of the failed nodeâĂŹs PGs had replicas that did not experience such a failure. As shown in the proof of correctness section, all correctly-behaving nodes have an overlapping good time, so if at least one PG replica behaved correctly, at least one would have prevented any writes from becoming visible (the issue covered in the problem discussion TODO REFERENCE). If there is at least one good node in each PG replica set, the snapshot is still safe.

Chapter 8

Future Work

With our research and analysis, we have also found points of interests should Red Hat or a future clinic team choose to pursue them. For one, the algorithm proposed can be used with any time synchronization protocol that provides guarantees on the clock uncertainty bounds. As such, alternative time protocols such as Chrony or the Precision Time Protocol (PTP) can be investigated to see if they could provide tighter uncertainty bounds than NTP. As of the time of this writing, those two particular protocols cannot provide guarantees on the uncertainty bounds, but could possibly be extended to include it and perhaps improve upon the bounds of NTP.

In addition, the algorithm proposed could be tested on a wider range of scenarios than was done for this project to prove its performance. The testing we did is more representative of smaller data centers with a single NTP server for synchronization. Other tests could include very large data centers, which would give more uncertainty on the network latency values and, more importantly, use more than one NTP server.

For our simulations, the model we currently use for clock drifting behavior is a consistently normal distribution. Unfortunately, we were unable to find literature describing better models for clock drifting behavior. With further analysis and testing, a better model could be found for clock drifting behavior to help analyze the performance of the behavior. The clock drifting behavior is very dependent on variables such as the Quartz crystal used, temperature and pressure variance, and changes due to the synchronization process.

Lastly, our algorithm still needs to be implemented and incorporated into Ceph. Our recommendations to approach this challenge are outlined in section 7. We believe that with a comfortable understanding of the Ceph infrastructure, our algorithm could be implemented without significant struggle.

Chapter 9

Conclusion

Through our research, analysis, and testing, we have found what we consider to be a satisfactory solution. As shown through our proof of correctness and testing, the algorithm allows for taking a consistent, point-in-time snapshot as desired from our problem statement. Even in the presence of out-of-band communication, the algorithm provides consistency among all of the files even without external dependency information. Our algorithm is at the level of each individual node, which means that a Ceph data center with this algorithm remains highly scalable without increasing impact on the system. Also, with reasonable guarantees on the uncertainty bounds provided by NTP, the freeze windows, and as a result the snapshot, should have negligible impact on a Ceph user. The algorithm is also robust to node failures since as long as one node in each PG remains up, the snapshot can be taken and the algorithm can succeed. We have found the NTP to be satisfactory for the usage in our algorithm in fulfilling the problem constraints, though We have also outlined alternatives and possible future work for using and testing other time synchronization protocols.

Appendix A

Team Plan

At beginning of the year, we proposed three possible plans of action to our liaisons:

1. *Viable Solution and Initial Integration*

   This would be the scenario where we had found a viable solution early enough in the academic year such that we would have had the time to both fully develop a final solution and to begin exploring how we would be able to integrate our solution into the current Ceph system. In this scenario, we would have found a viable solution sometime early in the fall semester. This would be after exploring a number of possible solutions and documenting why they were unsatisfactory. Our Midyear Report would contain details about what options we had explored, why they were unsatisfactory and a detailed explanation of what our solution was. It would also contain a proof that details why our solution is, theoretically, correct.

   The spring semester would be dedicated to creating a simulation data center that we could use to model an implementation of our solution. We would use a cluster of desktop machines or Raspberry Pis to simulate our solution to ensure that the constants on the runtime of our algorithm were not so large as to make our solution impossible to implement in real life. Finally, we would begin to look at the Ceph code base and see how we might be able to make recommendations about how to integrate our solution into Cephs system. Our Final Report would contain a final description of all of the work done over the course of the school year, including the description of the solutions we considered, the reasons why those solutions were dismissed, the final solution and a proof of its correctness, a description of a simulation of the solution, and recommendations for how to incorporate the solution into the current Ceph system.

2. *Viable Solution and Proof of Concept*

   This would be the case where we had found a viable solution later in the school year. While it would depend on when we finalized the solution, our Midyear Report would contain details about the algorithms that we had considered up to that point and a discussion about the drawbacks of the ones we discarded. Depending on when we had found a viable algorithm, we may have included in that document a description the algorithm and a proof of the correctness of that algorithm.

   The spring semester would be about proving that our solution is viable. We would create the proof of correctness, if we hadnâĂŹt already done it in the fall semester and we would create and run the simulations of our algorithm on an imitation cluster. Our Final Report would contain a final description of all of the work done over the course of the school year, including the description of the solutions we considered, the reasons why those solutions were dismissed, the final solution, a proof of its correctness and a description of a simulation of the solution.

3. *In-Depth Exploration of Ideas*

   The last scenario was the case where we are unable to find a solution. In that case, our goal would be to lay as much groundwork as we could for a future clinic team to take over after we finish. In the fall semester, we would do a significant amount of research into related work. We would try to apply the related work to come up with a solution and we would discuss all of the options that we considered and found were unsatisfactory. Our midyear report would contain all of the insights we gleaned over the semester, including the possible solutions that we explored and the reasons we decided they were unsatisfactory.

   In the spring semester, we would spend most of our time working on possible solutions, since we had spent most of the previous semester doing research. Our goal would be to come up with as many algorithms as we could and analyze the pros and cons of each of them. The Final Report would be devoted to describing the different algorithms and pieces of related work that we looked at over the year. We would also provide details into why the solutions that we looked at were unsatisfactory. The goal would be to create a document that provides a future clinic team with a good foundation to take up this project after us.

We completed the goals of the second plan.

Appendix B

Commonly Used Terms

B.1 General terms

- Cluster: computers that are networked together to perform a common task. A Ceph cluster is a bunch of computers that store data using Ceph.
- Node: a single computer in a cluster.
- Snapshot: a consistent recording of the entire state of a Ceph cluster
- Consistency: the partial ordering of events to preserve the causal relationships between them.
- Node Freeze: a hold on processing all writes that are sent to that node.
- Clock Drift: the natural drift over time that a computer clock experiences.
- Time Synchronization Protocol: a protocol that the nodes in a cluster implement to synchronize their clocks as they drift.
- Real Time: the true, global time of the cluster.
- Node Time: the time that the clock on a particular node reads; not guaranteed to be the same as real time.
- Out-of-Band Communication: communication between users of a Ceph cluster that Ceph has no way of detecting or recording.
- Node Failure: when a node in the cluster unexpectedly crashes, either temporarily or permanently.

B.2 Ceph specific terms

- RADOS (Reliable Autonomic Distributed Object Store): Object store service backing the Ceph file system.
- OSD (Object Storage Device): A given storage node within RADOS.
- PG (Placement Group): Aggregation of object groups for tracking object placement on OSDs within RADOS

Bibliography

2008. IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)* c1–269. doi:10.1109/IEEESTD.2008.4579760.

Burbank, Jack, David Mills, and William Kasch. 2010. Network time protocol version 4: Protocol and algorithms specification. URL https://tools.ietf.org/html/rfc5905.

Cardwell, N., S. Savage, and T. Anderson. 2000. Modeling TCP latency. In *IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, vol. 3, 1742–1751 vol.3. doi:10.1109/INFCOM.2000.832574.

Corbett, James C., Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's globally-distributed database. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, 251–264. OSDI'12, Berkeley, CA, USA: USENIX Association. URL http://dl.acm.org/citation.cfm?id=2387880.2387905.

Correll, Kendall, Nick Barendt, and Michael Branicky. 2005. Design considerations for software only implementations of the IEEE 1588 precision time protocol. *Conference on IEEE* 1588:11–15. URL http://dora.cwru.edu/msb/pubs/1588.pdf.

Fidge, Colin J. 1988. Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference*, vol. 10, 56–66.

Lamport, Leslie. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun ACM* 21(7):558–565. doi:10.1145/359545.359563. URL http://doi.acm.org/10.1145/359545.359563.

Lenzen, C., P. Sommer, and R. Wattenhofer. 2014. PulseSync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Transactions on Networking* Early Access Online. doi:10.1109/TNET.2014.2309805.

Lenzen, Christoph, Thomas Locher, and Roger Wattenhofer. 2010. Tight bounds for clock synchronization. *J ACM* 57(2):8:1–8:42. doi:10.1145/1667053.1667057. URL http://doi.acm.org/10.1145/1667053.1667057.

Lenzen, Christoph, Philipp Sommer, and Roger Wattenhofer. 2009. Optimal clock synchronization in networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 225–238. SenSys '09, New York, NY, USA: ACM. doi:10.1145/1644038.1644061. URL http://doi.acm.org/10.1145/1644038.1644061.

Maróti, Miklós, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. 2004. The flooding time synchronization protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, 39–49. SenSys '04, New York, NY, USA: ACM. doi:10.1145/1031495.1031501. URL http://doi.acm.org/10.1145/1031495.1031501.

Ongaro, Diego, and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, 305–320. USENIX ATC'14, Berkeley, CA, USA: USENIX Association. URL http://dl.acm.org/citation.cfm?id=2643634.2643666.

Sandeep Kulkarni, Murat Demirbas, Deepak Madeppa, Bharadwaj Avva, and Marcelo Leone. 2014. Logical physical clocks and consistent snapshots in globally distributed databases. URL http://www.cse.buffalo.edu/tech-reports/2014-04.pdf.

Weil, Sage A., Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. 2007. RADOS: A scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*, 35–44. PDSW '07, New York, NY, USA: ACM. doi:10.1145/1374596.1374606. URL http://doi.acm.org/10.1145/1374596.1374606.