

# Layered Interactive Convergence for Distributed Clock Synchronization

Raghukul Tilak, Alan D. George, Robert W. Todd

*High-performance Computing and Simulation (HCS) Research Laboratory*  
Department of Electrical and Computer Engineering  
University of Florida  
Gainesville, FL, 32611-6200

**Abstract** – A clock synchronization service ensures that spatially dispersed and heterogeneous processors in a distributed system share a common notion of time. In order to behave as a single, unified computing resource, distributed systems have need for a fault-tolerant, clock synchronization service. One approach employs the interactive convergence (ICV) method, which is both fully distributed and inherently fault-tolerant. However, this approach suffers from limits in terms of system scalability due to resource overhead. In this paper, a new layered form of ICV is introduced, compared with basic ICV and other simpler alternatives via experimental analysis on a distributed system testbed, and shown to improve synchronization tightness, resource utilization, and scalability.

**Index Terms** – Cluster computing, interactive convergence, clock synchronization, convergence, fault tolerance.

## 1. INTRODUCTION

With the ever-increasing performance levels and decreasing costs associated with uniprocessor and multiprocessor servers and desktop computers, high-performance cluster systems such as *Cplant* at Sandia National Laboratories hold the potential to provide a formidable supercomputing platform for a variety of grand-challenge applications. However, in order for heterogeneous and high-performance clusters to achieve their potential with parallel and distributed applications, a number of technical challenges must be overcome in terms of scalable, fault-tolerant computing. These challenges are made all the more complex with the increasing heterogeneity of technologies in network architectures, server architectures, operating systems, sharing strategies, storage subsystems, etc.

In distributed systems, one desirable capability is to have system-wide synchronized clocks, effectively extending approximately the same view of time-of-day throughout the system. A fault-tolerant clock synchronization service ensures that spatially dispersed processors in a distributed system share a common notion of time without being affected by the presence of faulty nodes and links in the system. Distributed applications can leverage such a service extensively for event ordering, distributed timeouts and measurement of elapsed time. In addition, such a service is essential for round-based applications such as deterministic gossip [1] and transaction-processing applications to be able to function in a fully distributed fashion. For example, round-based distributed applications function by keeping track of rounds using round numbers. To ensure correct functionality, the same round number should be seen on all processors at any given instance. However, providing system-wide consistent round numbers in turn demands system-wide synchronized clocks. Traditionally, distributed clock synchronization has been addressed using master-slave schemes, however such schemes suffer from an inherent lack of fault tolerance and scalability.

Fault-tolerant clock synchronization can be achieved in a fully distributed fashion using interactive convergence (ICV) algorithms [2]. Because of the distributed nature of ICV, each node interacts with all other

nodes in the system causing CPU utilization and network bandwidth utilization associated with ICV to be particularly high. This condition limits the scalability by imposing a limit on maximum system size that can be supported while keeping resource utilization within acceptable limits.

In this paper, a new hierarchical form known as *layered ICV* is introduced that provides improvements in the performance and scalability of conventional ICV and thus can support larger systems with comparable resource utilization. Experimental results are presented, analyzed, and compared from flat and layered implementations of ICV on a testbed consisting of a heterogeneous PC cluster comprised of one-hundred nodes connected by switched Fast Ethernet. As a basis for comparison, two master-slave [3-4] algorithms are also implemented. The performance of each clock synchronization service is measured in terms of the achievable synchronization tightness. Also, resource utilization is measured in terms of CPU utilization and network utilization.

The remainder of this paper is organized as follows. In the following section, a brief background on previous research and concepts related to clock synchronization is provided. The third section provides an overview of the design characteristics of the clock synchronization schemes implemented in the distributed testbed including the new layered form of ICV. Experimental results for different schemes are presented in the fourth section. Finally, the fifth section presents conclusions to the work presented and directions for future research.

## 2. BACKGROUND

For the last two decades, distributed clock synchronization has been studied extensively. However, most of the past research has focused on algorithmic aspects of the problem. Proposed clock synchronization algorithms in the literature can be broadly classified as *hardware-based* and *software-based* algorithms. Hardware-based algorithms [5-7] achieve very tight synchronization by employing specialized hardware like GPS receivers or time-stamping network interfaces. However, use of additional hardware makes these algorithms expensive to implement and inflexible. By contrast, this work's primary foundation is software-based algorithms [8-9], which use standard hardware but do not provide synchronization as tight as hardware-based algorithms. Work in [3] proposes a taxonomy for various synchronization schemes, and a survey of both hardware- and software-based approaches can be found in [10-12].

### 2.1 Basic Concepts

Clock synchronization algorithms can be used to synchronize clocks in a distributed system either among themselves or using a time reference external to the system. The first of the two methods, called *internal clock synchronization*, is effective for a system consisting of tightly coupled nodes. In this case, real time is not available within the system and the goal is to bring the maximum difference between any two clocks to a satisfactory minimum. The maximum difference between any two clocks achieved by the algorithm is called its *precision*. It is important to note that internally synchronized clocks, in spite of being very close to each other, might be inaccurate when compared to real time. The second method, called *external clock synchronization* [13], is useful for a system consisting of loosely coupled nodes. An external reference maintains the real time, and the goal is to keep all clocks

as close to the external reference as possible. The maximum difference between any of the clocks and the external reference that is achieved by the algorithm is called its *accuracy*.

Clock synchronization algorithms may be *asymmetric* or *symmetric* [3]. This property influences the algorithms's ability to support failures and determines its cost in terms of the number of messages exchanged to achieve the desired synchronization tightness. In asymmetric schemes, also known as *master-slave* schemes, nodes involved in clock synchronization play different roles. A specific role is assigned to a predefined node called the *master*. All other nodes in the system, called *slaves*, frequently use the master as a reference with which to synchronize [14-15]. The obvious advantage of asymmetric schemes is low resource usage because of their centralized nature. However, the master represents a single point of failure in the system. By contrast, in symmetric algorithms, each node plays the same role in the synchronization task. Symmetric algorithms are fully distributed and frequently use a communication step termed full-message exchange, during which each node communicates with all other nodes in the system resulting in high resource usage. The greatest benefit of symmetric schemes is their inherent support for fault tolerance with no single point of failure.

Asymmetric or master-slave algorithms can further be decomposed into *master-controlled* and *slave-controlled* schemes. In master-controlled schemes, the master node works as an active coordinator while the slave nodes work in a passive mode. The responsibility of initiating the resynchronization process periodically, after the elapse of a predefined time interval, lies with the master [16]. Slave-controlled schemes feature the master node in passive mode and slave nodes in active mode. Slave nodes initiate the resynchronization process periodically. A common example of a slave-controlled scheme is the *Network Time Protocol* (NTP) [13] used on the Internet for external synchronization, where slaves query the master periodically to obtain the reference time. Although master-slave schemes are naturally suitable for external clock synchronization, they can also be used effectively for internal clock synchronization. However, master-slave algorithms require extra mechanisms, such as fault detection followed by election of a new master, in order to recover from a failure of the master node. For example, NTP uses master replication where slaves use multiple synchronized masters in the resynchronization process.

## 2.2 Fault-Tolerant Clock Synchronization

The problem of keeping the clocks synchronized in a fault-tolerant fashion was first addressed by Lamport and Mellier-Smith in [17]. The ICV algorithm is one of the three algorithms that resulted from their work. ICV requires a fully-connected communication network among the nodes and handles arbitrary faults. The algorithm proceeds in rounds, synchronizing periodically to correct for clock drift. In a given round, each node obtains the clock values from each of the other nodes in the system. This sharing of clock values is accomplished by a full-message exchange among the nodes, requiring  $n(n-1)$  messages where  $n$  is the number of nodes in the system. Then, on each node, the collected remote-clock values are passed to an averaging function called the *convergence function*. This function reduces the input remote-clock values to produce a new value, which is then used to apply an offset to each node's local clock to achieve synchronization. The reduction process produces approximately the same clock value at each node and thereby achieves synchronization within some degree of precision.

The properties of the convergence function determine how fast synchronization can be achieved and whether the algorithm is resilient to failures or not. Some convergence functions may take several rounds before synchronization is achieved while others may take fewer. The fault-tolerance property of ICV is also attributed to the convergence function employed. For example, a convergence function may achieve fault tolerance by omitting from consideration remote-clock values from presumably faulty nodes during the reduction process, where faulty values are those found to be far outside the thresholds of clustered ones coming from healthy nodes. Overall, the convergence function is a critical element of the ICV algorithm.

Work presented in this paper uses the fault-tolerant midpoint averaging (FTMA) algorithm for convergence proposed in [2] that can sustain a maximum of  $f$  failures, provided the relationship  $n > 3f + 1$  holds true. FTMA works by discarding the  $f$  highest and  $f$  lowest remote clock values before finding the midpoint of the range of remaining values. The midpoint is used as a convergence point and synchronization is achieved by applying the calculated offset to the local clock at each of the nodes. The precision achieved using FTMA depends on the initial closeness of the clocks when the algorithm is started. There is an upper limit imposed on the initial closeness of the clocks which, if not met, causes FTMA to lose its convergence property. The performance of FTMA is experimentally measured and compared to other convergence functions in [18].

Several other properties of ICV make it suitable for distributed clock synchronization. ICV inherently caters to the heterogeneity prevalent in many distributed systems by making minimal assumptions about the underlying networks, network topologies and processing nodes. Moreover, because of its symmetric nature, ICV can easily be implemented as a system-level daemon. Such an implementation has been shown to improve performance in [19].

This paper uses the terminology proposed in [3] that is partially reproduced here. It is important to understand the view of time available to a distributed application. An application running on a node does not have access to the local hardware clock directly. Instead, it can access the *logical* clock maintained by the operating system using the underlying *physical* clock in the hardware. Thus, only logical clocks need to be synchronized. The precision achievable by a clock synchronization algorithm is a function of the maximum difference in the clock drift rates of any two logical clocks in the system. The maximum difference in the clock drift rates ( $\rho$ ) is a system property. The key design parameter involved in the clock synchronization service is the *resynchronization interval* ( $R$ ), which is the amount of real time elapsed between two consecutive events of resynchronization at a node in the system.

### 3. LAYERED ICV AND ALTERNATIVES

This section provides an overview of the basic design characteristics of clock synchronization algorithms that will be featured in the experimental testbed analysis. In addition to the layered ICV approach proposed in this paper, and the flat ICV on which it is based, two master-slave protocols are also implemented in the testbed and included here for the purpose of comparison.

#### 3.1 Master-Slave

Although the focus of this paper is on the design and analysis of a new layered form of ICV, two master-slave algorithms are first considered. As previously described, these algorithms (i.e. master-controlled and slave-

controlled) are included in the experiments as a baseline for assessing the performance of the new layered ICV algorithm.

The master-controlled scheme, shown in Figure 1(a), uses broadcast communication. The master executes a *server* process that has information about the current clock state. Each of the slaves executes a *client* process, which performs synchronization after collecting the required clock state information. The server process periodically broadcasts a clock synchronization message containing a time stamp based on its local clock. Client processes, on receiving the message, adjust their clock to the master's clock using the timestamp contained in the broadcast message. As the master node plays an active role in the process of disseminating the clock information, this scheme is referred to here as the *PUSH* scheme. This scheme relies primarily on the availability and reliability of broadcast communication. Unfortunately, by their very nature, scalable networks do not support hardware-based broadcast communication. Thus, the PUSH scheme may not be suitable for use in scalable systems.

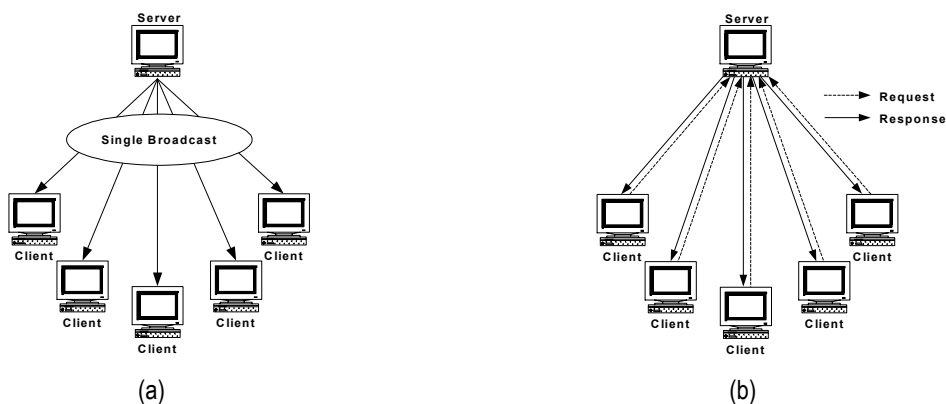


Figure 1. Master-controlled (a) and slave-controlled (b) schemes.

The scheme shown in Figure 1(b) is a slave-controlled scheme. As before, the server process executes on the master, while the slaves execute the client process. Clients periodically query the server process for the master's local clock value using a request packet. The server process responds with a running time stamp contained in the response packet. Clients estimate the master's clock by compensating the time stamp contained in the response packet with the communication delay between the master and slave nodes. Communication delay between the master and slave nodes is estimated to be half of the round-trip delay of the query message as measured by the client. This delay changes dynamically with traffic load on the network. The clients then adjust their clocks to the estimated value of the master's clock. As clients play an active role in acquiring the clock information from the master, this scheme is referred to here as the *PULL* scheme. In contrast to the master-controlled scheme, the transfer of clock information from the master to the slave nodes is accomplished using point-to-point communication. Delay compensation [18] is made possible by the two-way communication between the slave and master nodes. The slave-controlled scheme can tolerate large message-delay variations using delay compensation. As a result, this scheme performs well even in the presence of heavy traffic load on the network. By contrast, since the PUSH scheme uses one-way communication, and master and slave clocks are not yet synchronized, communication delay measurements may not be dependable. Hence, accurate delay compensation is not as easily achieved with the PUSH scheme.

Neither the PUSH nor the PULL scheme possesses the inherent resilience required for fault-tolerant systems, making symmetric synchronization techniques such as ICV potentially more suitable.

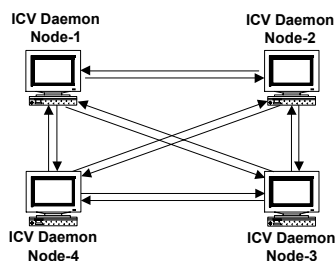


Figure 2. Full-message exchange of the interactive convergence (ICV) scheme.

### 3.2 Flat ICV

As previously described in Section 2, ICV imparts the functionality of both the server and the client on each of the nodes in the system. Figure 2 shows an ICV implementation for a four-node system. The ICV daemon is a multi-threaded implementation consisting of a *send thread* and a *receive thread*. The send thread and receive thread are functionally similar to server and client processes in master-slave schemes, respectively. Current clock state information is available to the send thread. Conversely, the receive thread receives clock state information from all other nodes in the system. The send thread initiates the synchronization process periodically by sending the current clock state information to all nodes in the system. It then sleeps for a predetermined duration allowing a sufficient amount of time for reception of clock state information from all other nodes. The send thread awakens after the duration and executes the ICV algorithm on the collected clock information. While the send thread sleeps, the receive thread can continue receiving clock state information from other nodes in the system. Instead of broadcast communication, only point-to-point communication is used for the exchange of clock information in order to support scalable networks.

### 3.3 Layered ICV

In its original form [17], ICV can be described as flat or unlayered ICV since it treats all nodes in the system as a single group. For example, consider the 16-node flat system shown in Figure 3(a). This system is comprised of a single group that contains all 16 nodes. ICV executes in this group and undergoes full-message exchange by sending and receiving  $n(n-1) = 16 \times (16-1) = 240$  messages during each resynchronization interval. Unfortunately, the scalability of flat ICV to large system sizes is severely limited by this inherent  $O(n^2)$  communication complexity. An alternative to flat ICV proposed here is layered ICV, where nodes in the system are divided into hierarchically arranged groups, with basic ICV executing within each of the groups. Tradeoffs are expected between flat and layered ICV in terms of simplicity and scalability of achievable precision and resource utilization, and these tradeoffs are explored in the experiments of the next section.

With layered ICV, a system of  $n$  nodes is divided into  $g$  groups or clusters, each containing  $m$  nodes, where  $n = m \times g$ . The nodes in each group operate using the basic form of ICV. The size and membership of groups may be selected to fit the underlying network topology. These groups form the bottom layer of the hierarchy. More groups

are formed by selecting nodes from the groups in the bottom layer. These groups are then hierarchically arranged to form two or more layers. For this work, two layers are assumed with layered ICV. Note that  $g$  represents the number of groups in the bottom layer in the hierarchy. The precision achieved by ICV for a given resynchronization interval depends on the maximum difference in the clock drift rates in the groups. Thus, groups may be formed so as to minimize the maximum difference in the clock drift rates within the groups, as permitted by the network topology. Since the upper-layer (L2) group is formed by selecting one or more nodes from each group in the lower layer (L1), L2 serves to connect the lower-layer groups. To ensure system-wide synchronization, all groups in a layer must have one or more nodes that are members of the upper-layer group as well.

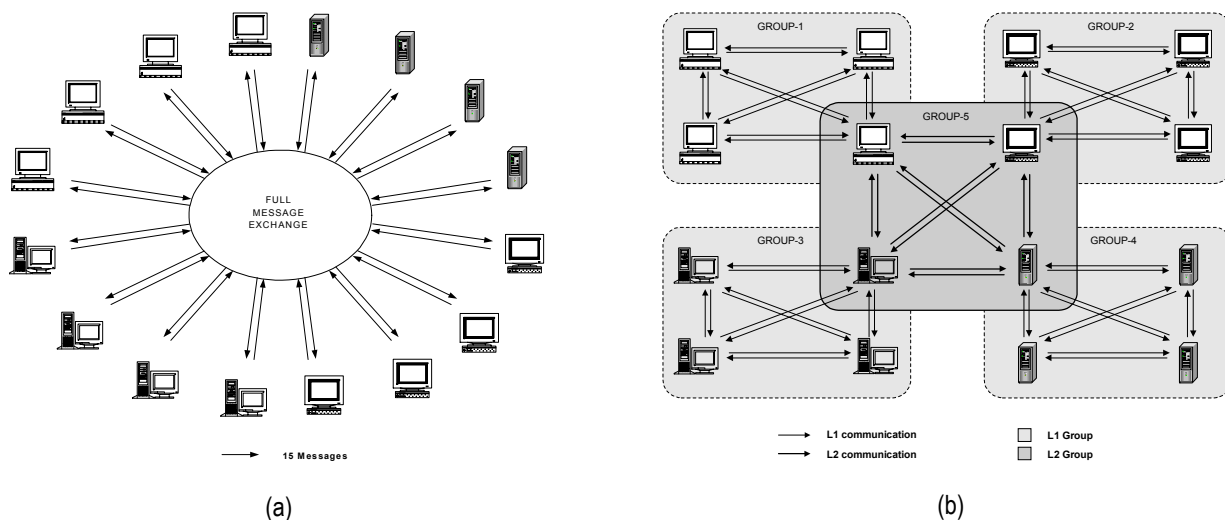


Figure 3. Interaction of (a) flat and (b) layered ICV in a 16-node heterogeneous system.

As an example, consider the 16-node system again. Layered ICV can be implemented by dividing the system into four ICV groups of four nodes each. These four groups encompass the lower layer. A group can then be formed at the upper layer by selecting one node (or more) from each of the lower-layer groups. Unlike the flat system that has a single ICV group, in this case the layered system has a total of five groups performing intra-group and inter-group ICV simultaneously. Such a system is shown in Figure 3(b).

Although presence of only one common node between L1 and L2 groups ensures connectivity, it also provides a single point of failure. Such a system is not fault-tolerant, as failure of the common node will isolate an L1 group from the rest of the system. Having more than one node from each lower-layer group participating in the upper-layer communication provides fault tolerance to the system. Using the FTMA algorithm cited in the previous section, flat ICV executing in a system of  $n$  nodes can sustain a maximum of  $f$  failures, where  $n > 3f + 1$ . For a group of  $m$  nodes in layered ICV, it follows that  $m > 3f + 1$ , and thus at the group level the maximum number of failures that can be tolerated is less than  $(m-1)/3$ . Therefore, maximum system-wide fault tolerance can be achieved by having  $(m-1)/3$  or more nodes involved in the L2 communication from each of the L1 groups.

In addition to fault tolerance, a large number of common nodes between L1 and L2 groups also provides better connectivity between the groups. Achievable precision is expected to improve with improved connectivity. Connectivity can also be improved by using small values of the resynchronization interval in the upper-layer group.

Layering may be used to improve the scalability of bandwidth and CPU utilization offered by the flat system. Per-node network consumption depends on the size of the ICV group. In contrast to a flat system, where all the nodes in the system form a single large group, groups in a layered system are smaller. With these tradeoffs in mind, we propose two options of layered ICV, one to achieve lower overhead and the other to achieve better precision.

Layering provides for smaller ICV groups, and a smaller group size results in reduced per-node network consumption. For the same reason, layering also results in reduced CPU utilization. This fact forms the basis of one layering strategy, termed *reduced bandwidth option*, which employs layering to achieve a reduction in per-node use of network bandwidth. However, this reduction may come at the expense of degraded precision when compared to a flat system of the same size, since nodes in one group have less precise information about the clock values of nodes in other groups than those within their group.

Alternately, achievable precision can be improved by resynchronizing more frequently while keeping per-node bandwidth consumption relatively unchanged. This strategy is termed the *improved precision option*, where layering is employed to achieve improved precision (versus a flat system of the same size) using approximately the same amount of resource utilization as that of the flat system.

Layering introduces two new parameters,  $R_1$  and  $R_2$ , which are resynchronization intervals in the L1 and L2 groups, respectively. The value of  $R_1$  dictates which of the two proposed layering alternatives is being used. With the reduced bandwidth option,  $R_1$  is kept the same as the resynchronization interval in the flat system  $R$ . By contrast, with the improved precision option,  $R_1$  is set to a value of  $R/g$ , and thus resynchronization frequency is increased.

The parameter  $R_2$  does not depend on the layering alternative being used. However, the value of  $R_2$  does depend on the maximum difference in the clock drift rates for the nodes selected to form the L2 group, and thus it has to be tuned. Ideally,  $R_2 < R_1$ . Although a smaller value of  $R_2$  ensures better connectivity between the L1 groups resulting in an overall tighter synchronization, it also results in increased resource consumption. For a given system,  $R_2$  should be set to an empirical value determined experimentally. The next section compares layered ICV systems based on these two layering options to several flat systems in terms of the resource utilization, achievable precision, and scalability.

## 4. EXPERIMENTS AND RESULTS

In this section, the results from precision, network utilization, and CPU utilization experiments on the distributed system testbed are presented. Along with layered ICV, all three of the traditional clock synchronization schemes previously introduced (i.e. PUSH, PULL, and flat ICV) are included for comparison.

Experiments were conducted on a heterogeneous PC cluster comprised of over one-hundred nodes. The cluster features a control network of switched Fast Ethernet. Each node contains an Intel IA-32 family processor executing the Redhat Linux V6.1/6.2 operating system with a V2.2 kernel. All clock synchronization algorithms

were implemented as standalone daemons on each node in the system. UDP was used as the transport protocol for exchange of clock synchronization messages providing low overhead and connectionless communication.

#### 4.1 Precision Experiments

In this set of experiments, the variation of precision with respect to  $\rho$ ,  $R$  and  $n$  is studied. The dependence of precision on each parameter is investigated by keeping the other two variables constant throughout each experiment. In each case, the worst precision (i.e. the highest value) observed during 100 resynchronization intervals is reported.

Measurement of precision was accomplished by sampling the clocks at all the nodes involved in the experiment and finding the maximum difference among the sampled values. Broadcast messages were used to signal the sampling instances to a high-priority process running on each node in the system. Acquired samples were compensated for differences in the message delay and protocol stack overhead before they were used to calculate the maximum difference. A predetermined offset was subtracted from the sampled clock value in order to minimize the measurement error introduced because of the differences in the message delay across the nodes. The control network that was used to carry the broadcast packets was not loaded, and thus no large variation in the message delay values was observed. Hence, the use of the statically determined offset values was apt. A dummy socket was employed to estimate the delay experienced by the broadcast packet in the protocol stack. An additional receive operation, which was performed on the dummy socket immediately after the receipt of the broadcast packet, was timed. The delay value obtained represents the time taken by the system call invoked to access the packet received from the network. This value was subtracted from the sampled clock value to reduce the measurement error due to the difference in the protocol stack delays across the nodes. As large variations in the protocol stack delay values were observed over a small period of time, the use of dynamically determined delay values was preferred. To ensure a reliable measurement, the precision value being measured was much larger than the broadcast message delay variation. The node used to signal the sampling instances was carefully selected based on the network topology in order to minimize the broadcast message delay variation.

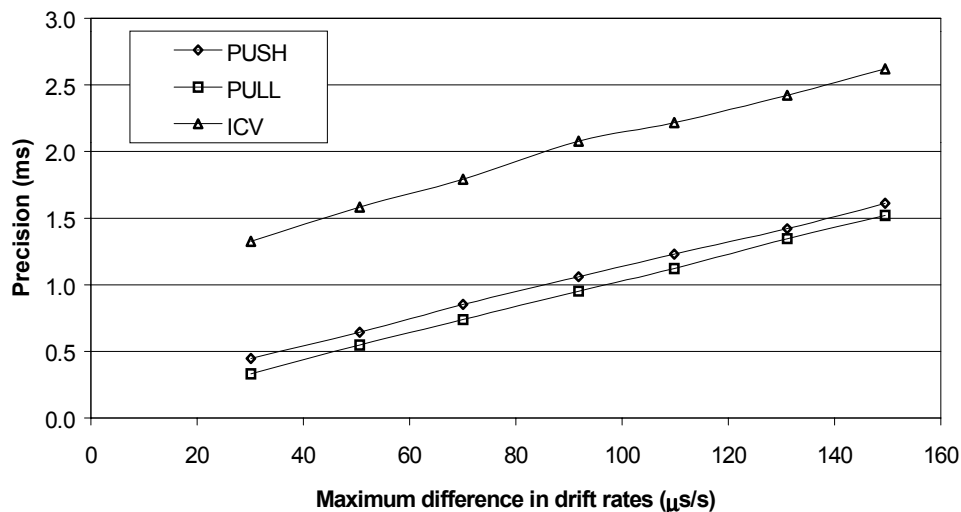


Figure 4. Variation of precision versus  $\rho$  for PUSH, PULL, and flat ICV ( $R = 10$  s,  $n = 32$ ).

In the first experiment, the effect of  $\rho$  on the precision is investigated. The composition of the 32-node group used in the experiment was changed to achieve different values of  $\rho$ . Figure 4 shows the precision values achieved by the three protocols for a 32-node system with the value of the resynchronization interval fixed at 10 seconds. It can be observed that precision increases linearly with  $\rho$  for all three protocols, exhibiting good scalability. Precision values achieved by PUSH are about 100 $\mu$ s higher than the corresponding values achieved by PULL for all values of  $\rho$ . The better performance of PULL can be attributed to the use of delay compensation. Although precision values achieved by ICV are the highest among the three protocols, thus showing poorest performance, ICV nevertheless is a strong contender because of its fault tolerance and comparable scalability.

In Figure 5(a), variation of the precision with resynchronization interval is presented. Precision exhibits a linear behavior with  $R$  for all protocols, showing good scalability. The precision value achieved by ICV is the highest among the three. Although PULL marginally outperforms PUSH, both the protocols scale in a similar fashion as evident from the slopes of the two curves in the figure. The slope of the ICV curve is slightly more than the other two curves, indicating slightly faster performance deterioration as compared to PUSH and PULL.

The effect of the system size on precision is shown in Figure 5(b). The PUSH and PULL curves remain flat with increasing values of  $n$ , thereby exhibiting ideal scalability in the region of interest. By contrast, the precision achieved by ICV shows  $O(n^2)$  variation with the system size. Thus, ICV performance degrades rather quickly with increasing system size.

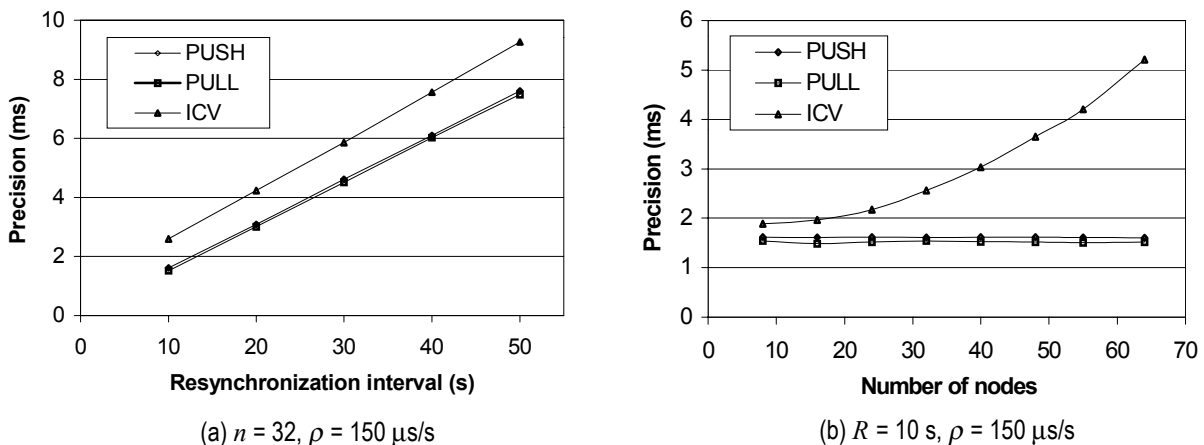


Figure 5. Variation of precision versus (a) resynchronization interval and (b) system size, for PUSH, PULL, and flat ICV.

Next, the performance of flat ICV is compared versus the two proposed forms of layered ICV. As before, a 32-node system with  $R = 10$ s was used to conduct the flat ICV experiments. For the layered protocols, the system was divided into groups such that the  $\rho$  value for each of the groups was minimum. Nodes were selected from the L1 groups to form the L2 group so as to keep the  $\rho$  in the upper-layer group small. To ensure the worst connectivity and thus the maximum (i.e. worst) possible value of precision, only one node from each of the L1 groups was used to form the L2 group. With the reduced bandwidth option, the system was divided into two 16-node groups. The resynchronization interval in the L1 groups ( $R_1$ ) was kept the same as the flat system ( $R$ ) at 10s. The layered system with the improved precision option was implemented by dividing the system into four 8-node groups. In order to

keep the per-node bandwidth consumption similar to the flat system,  $R_1$  was set to a value of 2.5 seconds. This value was obtained by downscaling the resynchronization interval for the flat system by a factor of four, which is the number of groups ( $g$ ) in the system. The resynchronization interval in the L2 group ( $R_2$ ) was determined experimentally so as to minimize overall precision achieved by the layered protocols. For both the layered protocols,  $R_2$  was varied while keeping all other parameters fixed and the achieved precision was measured. The precision values were observed to decrease with the reduction in  $R_2$  value. The decrease in the precision value was slow initially, followed by a rapid fall after which the slow rate of decrease was restored. The variation, thus, consisted of flat, steep and flat portions on the curve in that order, depicting the presence of a knee.  $R_2$  was set to an empirical value of 0.5 seconds which was the largest value after the knee of the experimental curve.

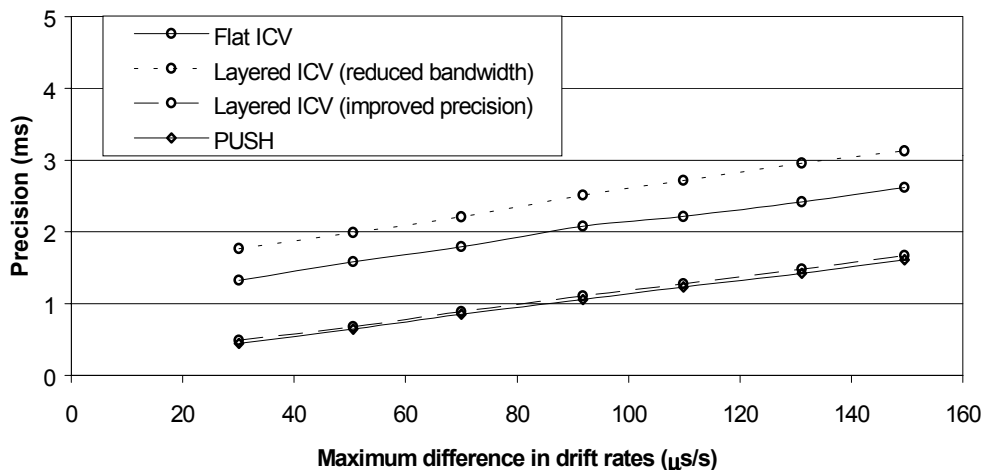


Figure 6. Comparison of flat and layered ICV in terms of achievable precision ( $R = 10$  s,  $n = 32$ ,  $R_2 = 0.5$  s; for reduced bandwidth option:  $R_1 = 10$  s,  $g = 2$ ; for improved precision option:  $R_1 = 2.5$  s,  $g = 4$ ).

A comparison between the flat and layered systems is shown in Figure 6. The layered protocols show linear variation with  $\rho$ , exhibiting scalability similar to that of the flat system. The layered system with the reduced bandwidth option resulted in an approximate increase of 30% over the value of the precision achieved by the flat system. This increase can be attributed to the fact that nodes in one group do not have precise information about the clock values of the nodes in the other groups. By contrast, the layered system with the improved precision option achieved a reduction of approximately 40% to 60% in the values of precision recorded compared to the flat system. This reduction can be attributed to the use of a smaller value of the resynchronization interval in the layered protocol. It is noted that layered ICV with the improved precision option performs similar to the PUSH protocol. Thus, layering can be employed to overcome the disadvantage related to the poor performance of flat ICV as compared to the PUSH and PULL protocols. This observation is important because, in spite of the desirable properties of being fully distributed and fault-tolerant, the usefulness of flat ICV is limited due to its poorer performance.

## 4.2 Network Utilization Experiments

The network bandwidth utilization of the distributed clock synchronization service was measured on a per-node basis using *Etherreal*, a network analyzer available in the public domain. The network bandwidth associated with the clock synchronization daemon was measured by capturing the synchronization message packets generated by a node over a period of time and then estimating the network bandwidth consumed. Network utilization of the clock synchronization process/daemon is a function of the clock synchronization scheme, system size, and resynchronization interval. Unlike the precision experiments, the resynchronization interval was set to a value of 1 second, typically the smallest value of the resynchronization interval. Network bandwidth utilization results for the three protocols was measured for a system of up to 96 nodes. Network utilization for the asymmetric master-slave schemes, PUSH and PULL, was measured separately for the client and the server processes.

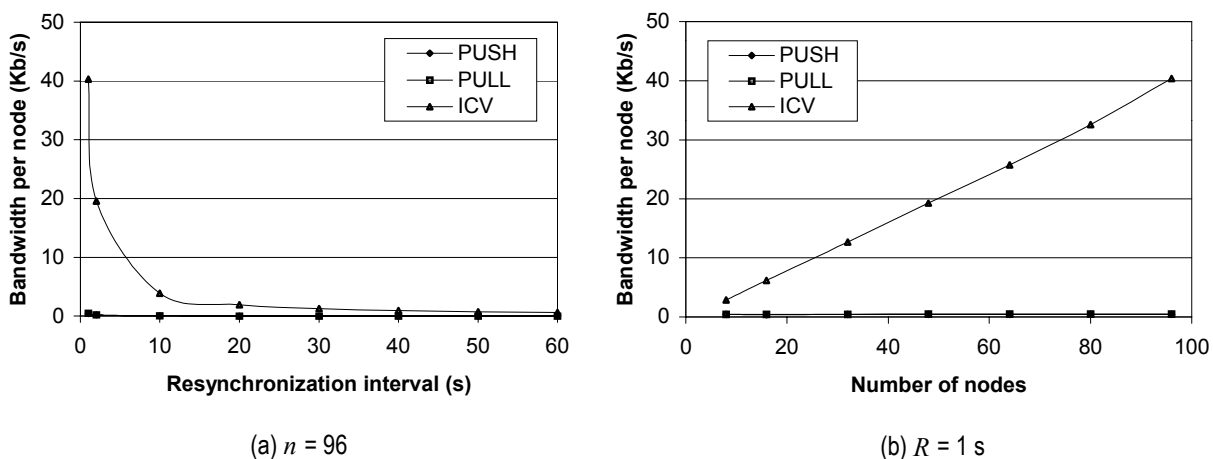


Figure 7. Variation of client network utilization versus (a) resynchronization interval and (b) system size, for PUSH, PULL, and flat ICV.

The dependence of per-node network bandwidth utilization on the resynchronization interval for the client process is shown in Figure 7(a). Since the receive thread in the ICV daemon is functionally similar to the client process, the bandwidth utilization associated with it has also been included for comparison. Per-node network bandwidth usage increases hyperbolically with decreasing  $R$  for all three schemes. Such a variation makes per-node client network utilization critical for small values of  $R$ . Due to the large difference in the magnitudes of the ICV and PUSH or PULL curves, the latter two are not clearly visible in the plot. Such a large difference in magnitudes is due to the fact that the per-node network utilization for the receive thread in ICV scales with system size as  $O(n)$  as shown in Figure 7(b). Dependence on the system size occurs since the receive thread communicates with all nodes in the group, unlike the PUSH and PULL clients where per-node network bandwidth utilization is independent of system size and thus they exhibit ideal scalability.

The variation of the per-node network bandwidth utilization for the server process with respect to the resynchronization interval is shown in Figure 8(a). Owing to the functional similarity of the send thread to the server process, the bandwidth associated with the send thread in the ICV daemon is also included in the graphs. For all three protocols, per-node network bandwidth utilization increases hyperbolically with decreasing  $R$ . The

relatively small magnitude of the PUSH curve makes it appear flat in the graph. It is observed that per-node network bandwidth consumed by the PULL server is almost twice the amount consumed by the send thread in ICV. Such behavior occurs because the PULL protocol uses two-way communication as compared to one-way communication used by ICV. Network utilization scalability for the three protocols is shown in Figure 8(b). Here, both ICV and PULL scale as  $O(n)$ , whereas PUSH does not depend on system size. Figure 8(b) also indicates that the PULL server consumes twice the amount of bandwidth consumed by the send thread in ICV.

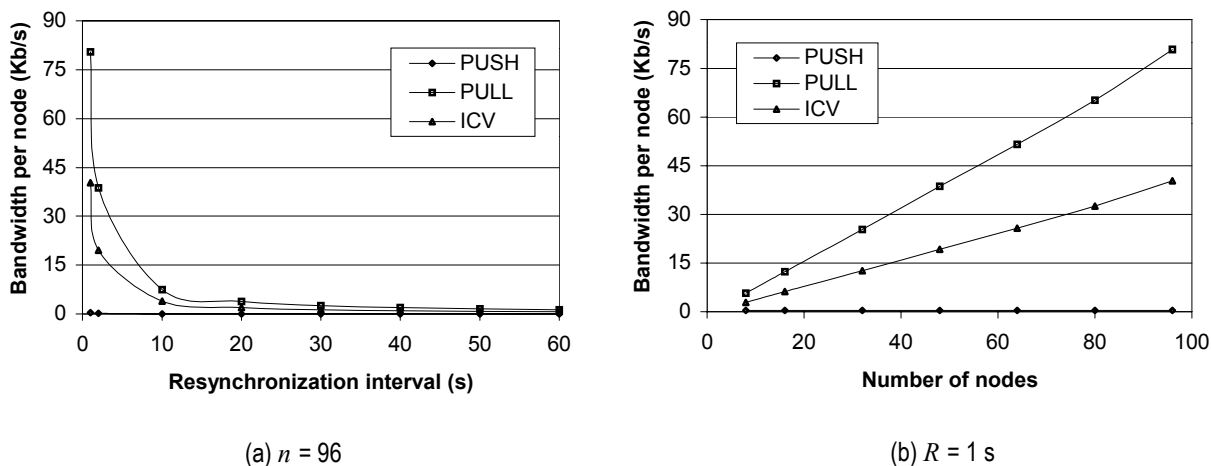


Figure 8. Variation of server network utilization versus (a) resynchronization interval and (b) system size, for PUSH, PULL, and flat ICV.

Because of the relatively higher network utilization as compared to the client processes, the node on which a PULL server functions may become a “hot spot” in the network. Also, the scalability of the PULL protocol may be limited by the network bandwidth consumption of the server process.

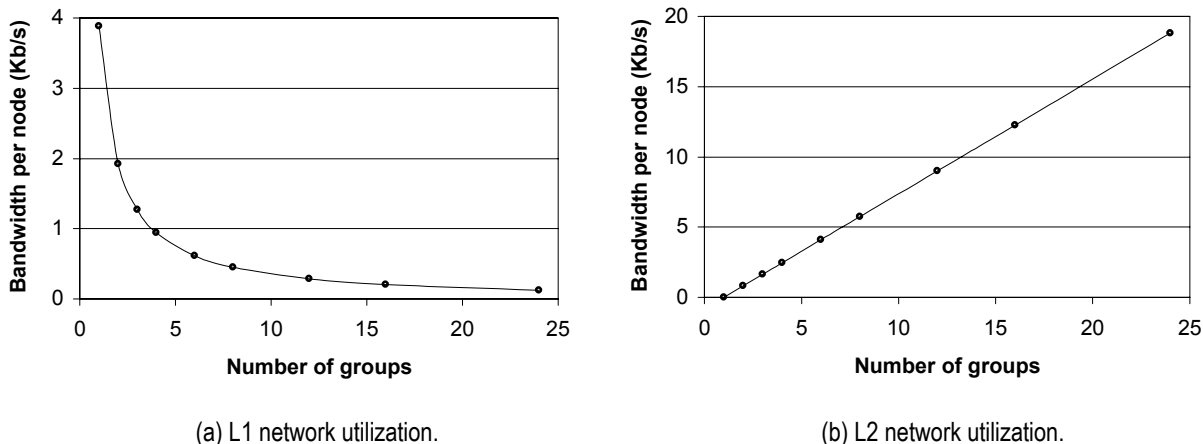


Figure 9. Impact of number of groups on network utilization in layered ICV with reduced bandwidth option ( $n = 96$ ,  $R_1 = 10$  s,  $R_2 = 0.5$  s).

Network utilization for a layered ICV system was investigated by dividing a 96-node system into two layers as described previously. Both the reduced bandwidth and improved precision options were included with a setup

similar to the one used in the previous layering experiments. The network utilization for the L1 and L2 communication was measured separately. Finally, the total per-node network utilization was found by adding the L1 and L2 contributions. Figures 9(a) and 9(b) show the L1 and L2 per-node network utilization, respectively, for the reduced bandwidth option with a varying number of groups in the lower layer.

For a fixed system size, the L1 per-node network utilization decreases hyperbolically with an increase in the number of groups. Such a behavior is justified as the number of nodes in an L1 group decreases with an increase in the number of groups for a given system size. The L2 per-node bandwidth utilization exhibits  $O(g)$  scalability since the number of nodes in the L2 group increases linearly with the number of L1 groups in the system. The total per-node network bandwidth obtained by adding the L1 and L2 contributions is shown in Figure 10.

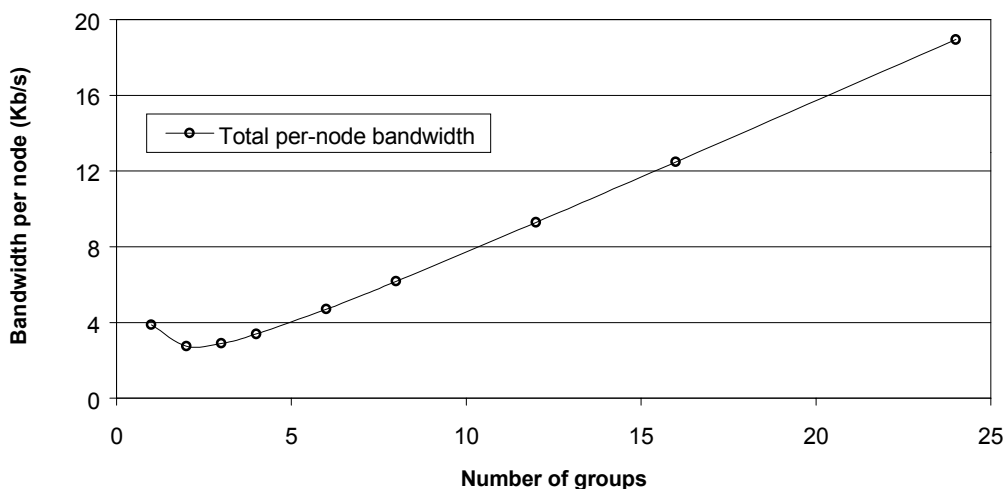


Figure 10. Impact of number of groups on total per-node network utilization ( $n = 96$ ,  $R_1 = 10$  s,  $R_2 = 0.5$  s).

In Figure 10, it is observed that the total per-node network utilization decreases initially with the increase in the number of groups, attains its minimum value, and increases linearly thereafter. Thus, there exists an optimum number of groups that minimizes the per-node network utilization in a layered system. Existence of such a minimum is expected as L1 per-node bandwidth utilization decreases while L2 per-node bandwidth utilization increases, with an increase in the number of groups in the system. The value of  $g$  at which the minimum occurs depends on the rate at which the L1 contribution decreases and the rate at which the L2 contribution increases. For the system under consideration, dividing the system into two groups optimizes the per-node network utilization.

The improved precision option tends to maintain a constant L1 per-node bandwidth by reducing the value of the resynchronization interval with each increase in the number of groups in the system. Thus, by contrast, for the improved precision option there does not exist an optimum number of groups that minimizes per-node network utilization. It was observed that the total per-node network utilization with the improved precision option increases monotonically with an increase in the number of groups in the system.

The scalability of the aggregate network utilization of the flat and layered systems is compared in Figure 11. The same system setup was used as in the previous layering experiments. While the aggregate bandwidth utilization

curves for the flat system and the layered system with the improved precision option both exhibit  $O(n^2)$  scalability, the aggregate bandwidth utilization of the layered system with the reduced bandwidth option shows  $O(n^2/g)$  scalability. The improved precision option achieves a smaller precision value while consuming the same aggregate bandwidth as the flat system. It may be noted that, for small system sizes, the layered system with the improved precision option has higher aggregate network utilization than that of a flat system because of the added L2 contribution, which is substantial when the L1 contribution from a small number of nodes is insignificant. For larger system sizes, the fixed L2 contribution is amortized over a larger number of nodes resulting in a smaller aggregate network utilization for the layered system.

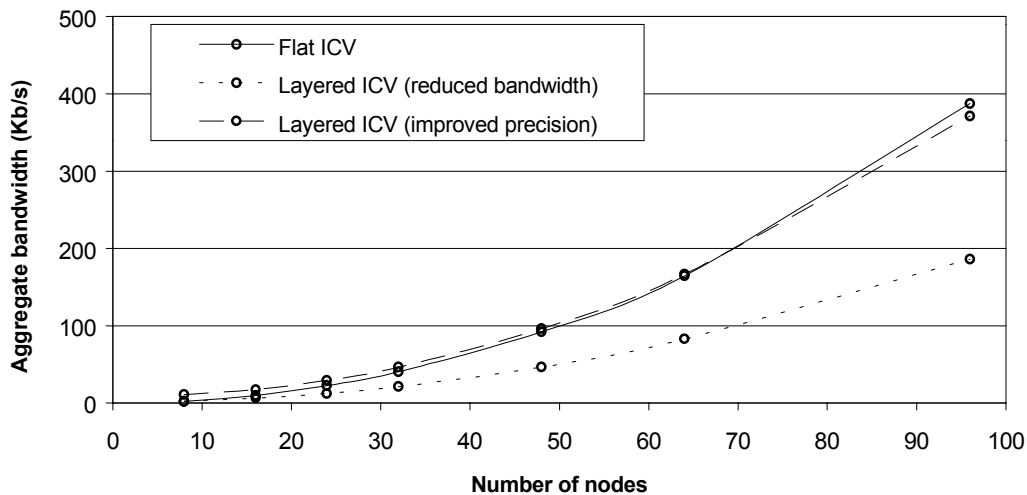


Figure 11. Scalability comparison of aggregate network utilization in flat and layered ICV systems ( $R = 10$  s,  $n = 32$ ,  $R_2 = 0.5$  s; for reduced bandwidth option:  $R_1 = 10$  s,  $g = 2$ ; for improved precision option:  $R_1 = 2.5$  s,  $g = 4$ ).

### 4.3 CPU Utilization Experiments

CPU utilization for each clock synchronization service was estimated by measuring the number of CPU cycles consumed during a resynchronization interval. CPU utilization values averaged for over one-hundred resynchronization intervals are reported. Three different node architectures with different CPU configurations were used to conduct the experiments, the results of which are presented as a family of curves. The three node architectures used are summarized in Table 1. The setup used in this set of experiments was similar to the one used in the network utilization experiments.

Table 1. Summary of node architectures.

Node type	Configuration
Alpha	400MHz Intel Celeron PPGA processor with integrated 128KB L2 cache
Delta	600MHz Intel Pentium-III processor with 512KB L2 cache
Zeta	733MHz Intel Pentium-III processor with integrated 256KB L2 cache

CPU utilization for the PUSH and PULL clients is shown in Figure 12. Because of a great difference in the magnitudes of the PUSH or PULL and ICV, processor utilization for ICV is shown separately. In the experimental results, it was observed that the Alpha, Delta, and Zeta nodes exhibited similar trends with only an insignificant vertical translation between them. Thus, only results with Zeta nodes are presented for PUSH and PULL. As shown in Figure 12(a), CPU utilization for both PUSH and PULL experiences a hyperbolic decrease with an increase in  $R$ . Because of this behavior, CPU utilization becomes of critical importance for small values of  $R$ . It was observed that for a resynchronization interval less than the operating system time slice (i.e. 10ms for Linux), processor utilization is approximately 100% because the clock synchronization service has to busy-wait on the CPU to achieve an accurate value of the resynchronization interval. Although the PUSH scheme is the most economical from the point of view of CPU resource usage, it has less applicability, as most scalable networks do not support a true broadcast. In Figure 12(b), the scalability of CPU utilization for the client processes is shown. CPU utilization for the PUSH and PULL schemes exhibit ideal scalability, showing no dependence on the system size. This behavior is due to the fact that addition of more nodes in the system only affects the server and not the other clients. Because of two-way communication, the PULL client consumes more CPU resources than PUSH for a similar configuration.

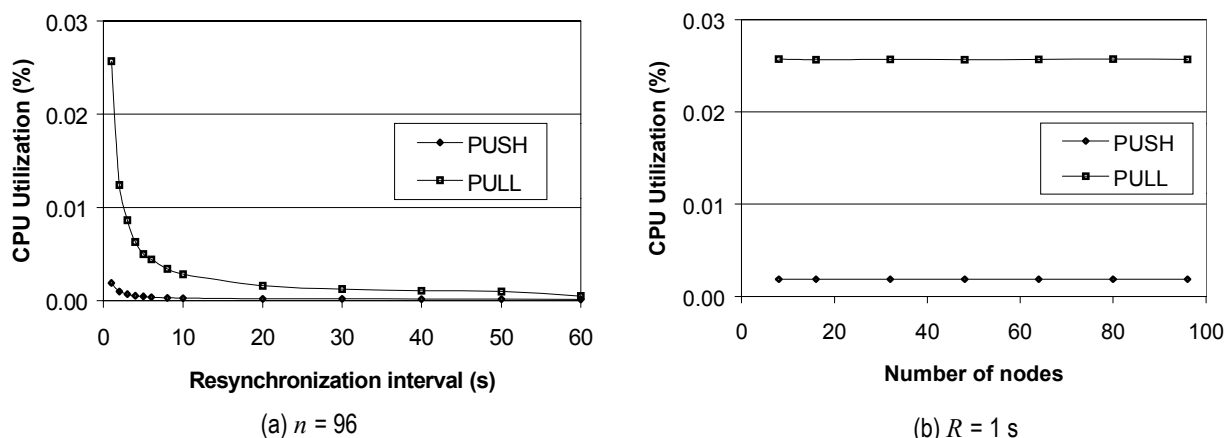


Figure 12. Variation of client process CPU utilization versus (a) resynchronization interval and (b) system size, for PUSH and PULL.

For the server process, the dependence of CPU utilization on resynchronization interval and system size is depicted in Figure 13. Again, only results for Zeta nodes are presented. Similar to the client process, CPU utilization for the server process decreases hyperbolically with increasing  $R$ . In Figure 13(a), it can be observed that there is a significant difference in the magnitudes of the PUSH and PULL curves, because of which, the PUSH curve appears to be flat. This difference can be attributed to the fact that the CPU utilization for the PULL server depends on the system size, and in a 96-node system large differences in the magnitude are justified. This difference grows with increasing system size. This observation is confirmed in Figure 13(b) which shows  $O(n)$  scalability for CPU utilization for the PULL server. Unlike the PULL client, the dependence of processor utilization for the PULL server on the system size may limit the scalability of the protocol. CPU utilization for the PUSH server exhibits ideal scalability, showing no dependence on the system size. Since the PUSH server uses broadcast communication

in contrast to the point-to-point communication used by the PULL server, the addition of more nodes in the system does not burden the PUSH server.

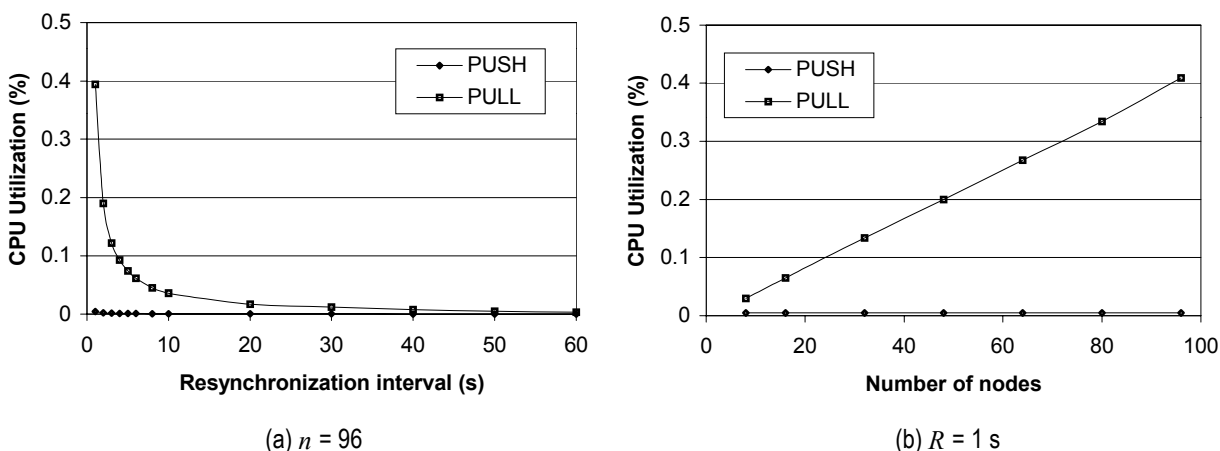


Figure 13. Variation of server process CPU utilization versus (a) resynchronization interval and (b) system size, for PUSH and PULL.

Figure 14 shows the effect of the resynchronization interval and system size on CPU usage for flat ICV. As ICV is more intensive on CPU resources, results for all three types of node architectures are shown. In the region of interest, processor utilization shows a hyperbolic decrease with increasing resynchronization interval. For the same reason as detailed before, CPU utilization will reach 100% if the value of  $R$  is chosen to be less than the operating system time slice. From Figure 14(a), for a 96-node flat system resynchronizing every second, the ICV daemon used approximately 1.9% of the CPU resource on an Alpha node. CPU utilization for the flat ICV scales as  $O(n^2)$  in Figure 14(b), and thus is expected to exceed acceptable limits quickly as the system grows in size.

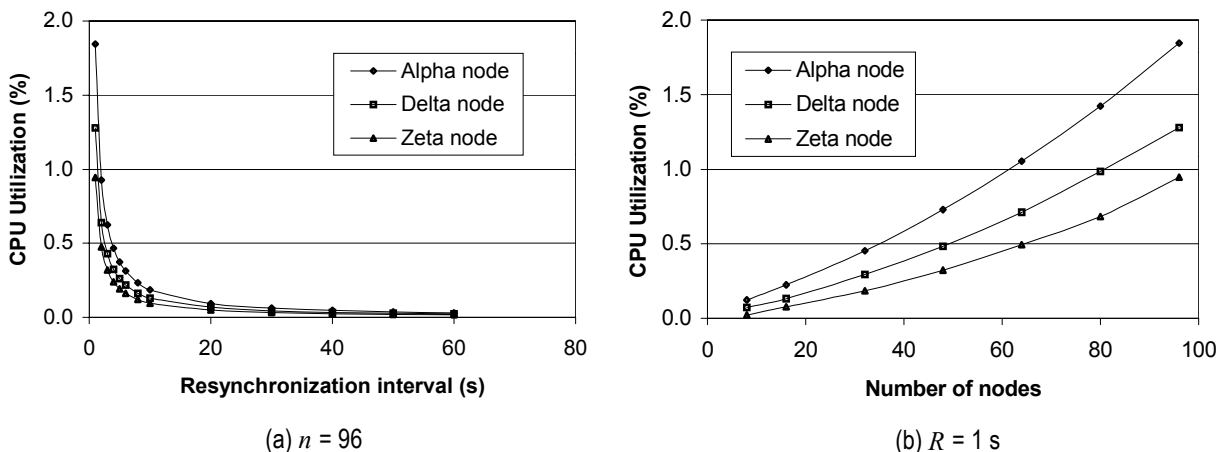


Figure 14. CPU utilization versus (a) resynchronization interval and (b) system size, for flat ICV.

To measure the processor usage of the layered system, a setup similar to the one employed in the network utilization experiments was used. The L1 and L2 components of processor utilization for a layered system were

measured separately and added to find the total processor utilization. Figure 15 shows the L1 and L2 components for the three types of nodes using the reduced bandwidth option. In Figure 15(a), L1 processor utilization is observed to experience a parabolic decrease with increasing number of groups for a fixed system size. This behavior is due to the fact that the number of nodes in the L1 group decreases with an increase in the number of groups, for a fixed system size. The variation of L2 processor utilization is depicted in Figure 15(b). Given the scalability of flat ICV, for a system partitioned into two or more groups it follows that the CPU utilization of L2 in layered ICV has  $O(g^2)$  scalability. With just one group, the system becomes flat, causing the L2 processor utilization to become zero.

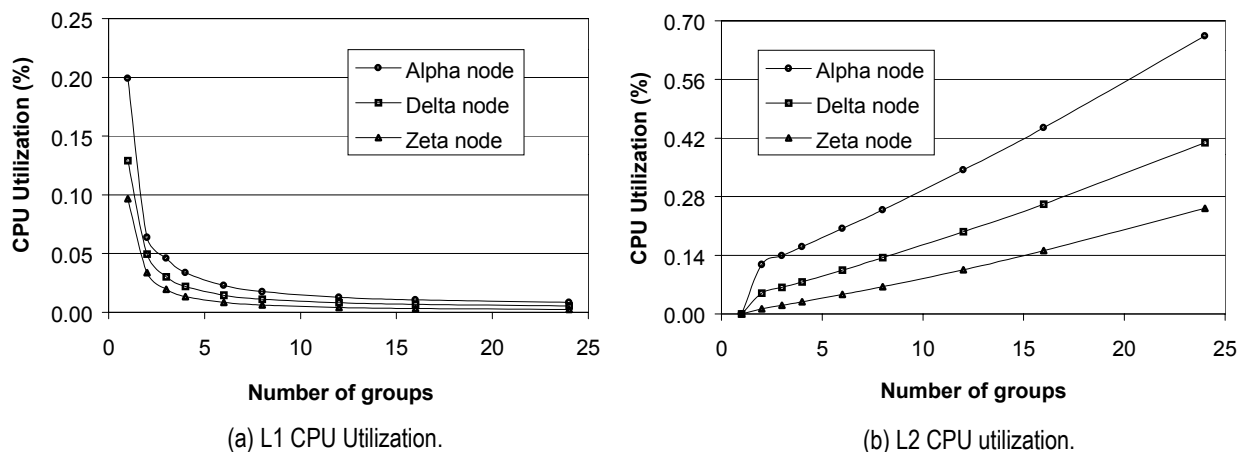


Figure 15. Effect of number of groups on CPU utilization in layered ICV ( $n = 96$ ,  $R_1 = 10$  s,  $R_2 = 0.5$  s).

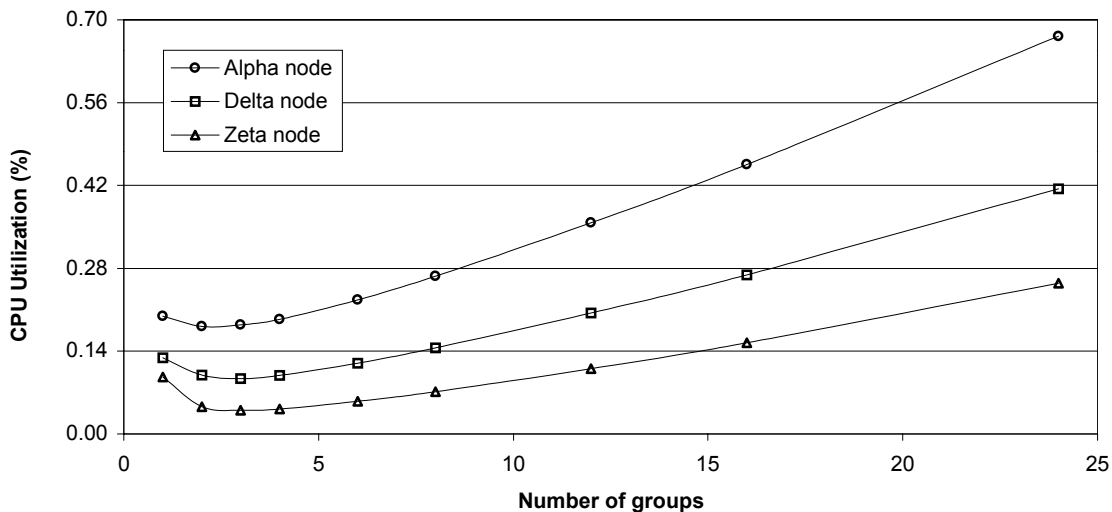


Figure 16. Impact of number of groups on total CPU utilization in layered ICV ( $n = 96$ ,  $R_1 = 10$  s,  $R_2 = 0.5$  s).

The total processor utilization for the layered system with the reduced bandwidth option is presented in Figure 16. Because of the opposite behaviors of L1 and L2 components with an increase in the number of groups in the system, the total processor utilization experiences an initial decline followed by a rapid increase. Thus, for a fixed

system size, there exists an optimum number of groups that minimizes the total CPU utilization in the layered system. As shown in Figure 16, the optimum number of groups for a 96-node system is 2.

It is important to note that the optimum number of groups to minimize network utilization and to minimize processor utilization may not be the same. If so, a layered system can thus be designed to minimize either network or processor utilization, but not both. By contrast, in a layered system with the improved precision option, the L1 component remains constant while the L2 component grows with the number of groups for a given system size. Thus, the total processor utilization rises monotonically with an increase in the number of groups, and there is no optimum value for the number of groups that will minimize the total processor utilization.

Finally, scalability of the flat and layered systems is compared in terms of the CPU resource usage in Figure 17. Layering in general results in reduced CPU resource consumption. For a 96-node system, layered ICV with the reduced bandwidth option consumes only 35% of the CPU resources required by flat ICV. This improvement in the scalability can be attributed to the fact that in a layered system each node communicates only with the other nodes in its group. By contrast, a layered system with the improved precision option consumes about 50% of the CPU resources consumed by the flat ICV.

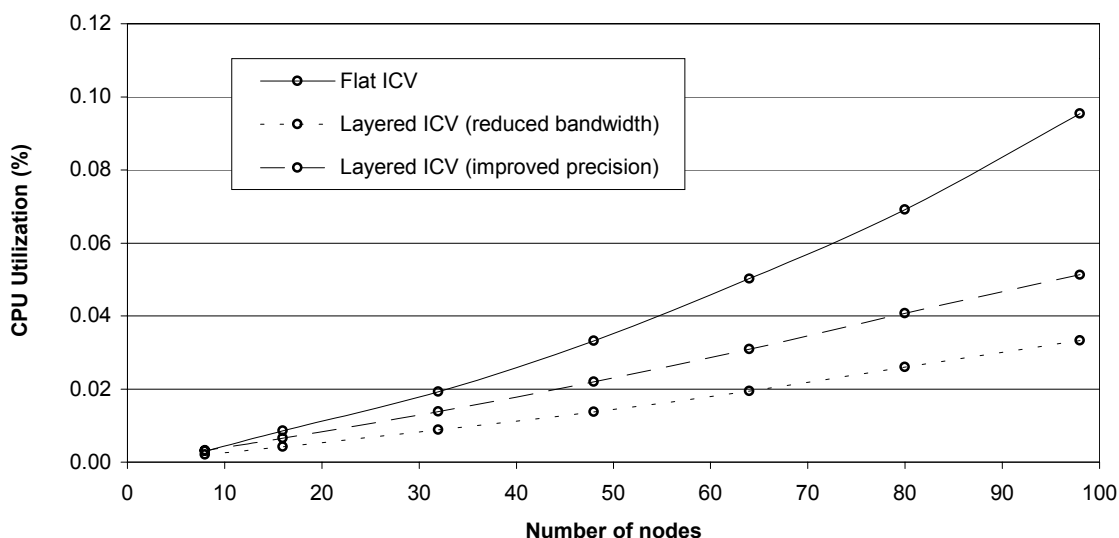


Figure 17. Scalability comparison of CPU utilization in flat and layered ICV ( $R = 10$  s,  $n = 32$ ,  $R_2 = 0.5$  s; for reduced bandwidth option:  $R_1 = 10$  s,  $g = 2$ ; for improved precision option:  $R_1 = 2.5$  s,  $g = 4$ ).

## 5. CONCLUSIONS

This paper proposes a layered form of ICV for system-wide clock synchronization as an alternative to flat ICV in order to improve performance and scalability of the latter. A comprehensive performance analysis of the proposed method along with a comparison versus several alternatives is presented based on experiments with a cluster testbed of 100 nodes. The master-slave clock synchronization schemes (PUSH and PULL) are used as a basis to evaluate the performance of ICV. The performance and scalability of both flat and layered systems is compared and

contrasted in terms of achievable synchronization tightness, the level of network bandwidth utilization, and the level of processor utilization.

For all clock synchronization schemes, precision scales linearly with the maximum difference in drift rates keeping resynchronization interval fixed and vice-versa. Among the master-slave schemes, the PULL protocol is found to outperform the PUSH protocol marginally by achieving precision values  $100\mu\text{s}$  less than the latter. The better performance of the PULL protocol is attributed to the use of delay compensation. Although flat ICV has the poorest performance of these three in terms of achievable precision, it is more applicable than the master-slave schemes because of its fault-tolerance properties.

Layering is shown to improve the precision achieved by ICV. Layered ICV systems with the improved precision option, which achieved 40% to 60% better precision than the flat system using the same network bandwidth resource and 50% less CPU resources, are comparable to the master-slave schemes in terms of achieved precision and scalability. Layering with the reduced bandwidth option resulted in 30% worse precision than the flat system but the network utilization and processor utilization were reduced to 50% and 35% that of the flat system, respectively.

Although, from the resource utilization point of view, the PUSH scheme is the most economical, it has limited applicability because it relies on the availability and reliability of broadcast. In flat ICV, per-node network utilization was found to show  $O(n)$  scalability while the processor utilization exhibited  $O(n^2)$  scalability. Network bandwidth utilization per node of a flat system is minimal (e.g.  $\cong 40\text{Kb/s}$  for a 96-node system resynchronizing every second) owing to the small size of the clock synchronization messages. Because of the second-order variation, the processor usage grows rather quickly with system size, thus limiting scalability of flat ICV.

Layering is also shown to improve the scalability of ICV by cutting down the resource utilization associated with it. Aggregate bandwidth utilization for a layered system with the reduced bandwidth option scales as  $O(n^2/g)$  in contrast to a layered system with the improved precision option which scales as  $O(n^2)$ . Processor usage per-node of the layered systems with the reduced bandwidth and improved precision options is about 35% and 50% of the processor utilization of the flat system, respectively.

In summary, the proposed approach of layered ICV has proven to provide precision, resource overhead, and scalability characteristics competitive with the master-slave methods. In so doing, it maintains the many inherent advantages of ICV in terms of symmetry, point-to-point communication, no single point of failure, no hot spot, resilience in the presence of failures, etc.

Several directions for future research may be considered. For instance, while this research focused on hierarchical systems with two layers, for very large systems three or more layers may be beneficial, and thus the need exists for studies such as scalability and tradeoff comparisons in terms of resource utilization and achievable precision. Another direction relates to how information is communicated. The payload size of the clock synchronization messages is typically very small, in fact smaller than the size of the header in the transmission frames. Thus, sending clock synchronization messages as separate packets is not particularly efficient with available network bandwidth. One method of sending the clock state information in a more efficient way may be to “piggy-back” the clock synchronization messages onto packets generated by other distributed system-level services.

Finally, further studies are needed on the effects of various layering strategies on system-wide fault tolerance, perhaps using other convergence functions to improve resilience such as the Sliding Window Averaging (SWA) function found in [18].

## ACKNOWLEDGEMENTS

The support provided by Sandia National Labs on contract LG-9271 is acknowledged and appreciated, as are equipment grants from Nortel Networks, Intel and Dell that made this work possible.

## REFERENCES

1. Ranganathan, S., George, A., Todd, R. and Chidester, M. "Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters," *Cluster Computing*, 4(3):197-209, July 2001.
2. Lundelius, J. and Lynch, N. A. "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, 1988, 77:1-36.
3. Anceaume, E. and Puaut, I. "Performance evaluation of clock synchronization algorithms," Research report INRIA, #RR3526, October 1998.
4. Cristian, F. "Probabilistic clock synchronization," *Distributed Computing*, 3:146-153, 1989.
5. Shin, K. G. and Ramanathan, R. "Clock synchronization of a large multiprocessor system in presence of malicious faults," *IEEE Transactions on Computers*, C-36(1):2-12, 1987.
6. Kopetz, H. and Ochsenreiter, W. "Clock synchronization in distributed real-time computer systems," *IEEE Transactions on Computers*, C-36(8):933-940, August 1987.
7. Schossmaier, K. "An interval-based framework for clock rate synchronization," *Proceedings of the 16<sup>th</sup> ACM Symposium on Principles of Distributed Computing (PODC)*, Santa Barbara, USA, August 21-24, 1997.
8. Verissimo, P., Casimiro, A. and Rodrigues, L. "Cesiumspray: a precise and accurate global time service for large scale systems," *Journal of Real-Time Systems*, 12:243-294, 1997.
9. Fetzer, C. and Cristian, F. "Integrating external and internal clock synchronization," *Journal of Real-Time Systems*, 12(2): 123-172, 1997.
10. Schneider, F. B. "A paradigm for reliable clock synchronization," Technical Report TR86-735, Computer Science Department, Cornell University, February 1986.
11. Ramanathan, P., Shin, K. G. and Butler, R. W. "Fault-tolerant clock synchronization in distributed systems," *IEEE Computer*, 23(10):33-44, October 1990.
12. Simons, B., Welch, J. L. and Lynch, N. "An overview of clock synchronization," *Asilomar Workshop on Fault-tolerant Distributed Computing Conference*, Vol. 448, pp. 84-96, *Lecture Notes in Computer Science*, 1990.
13. Mills, D. L. "Internet time synchronization: the network time protocol," *IEEE Transactions on Computers*, 39(10):1482-1493, October 1991.
14. Cristian, F. "Probabilistic clock synchronization," *Distributed Computing*, 3:146-158, 1989.
15. Arvind, K. "Probabilistic clock synchronization in distributed systems," *IEEE Transactions in Parallel and Distributed Systems*, 5(5):474-487, May 1994.
16. Gusella, R. and Zatti, S. "An election algorithm for a distributed clock synchronization program," *Proceedings of 6th International Conference on Distributed Computing Systems*, pp. 364-373, 1986.
17. Lamport, L. and Melliar-Smith, P. M. "Synchronizing clocks in the presence of faults," *Journal of the ACM*, 32(1):52-78, January 1985.
18. Azevedo, M. M. de and Blough, D. M. "Fault-tolerant clock synchronization for distributed systems with high message delay variation," in *Fault-Tolerant Parallel and Distributed Systems* (D. Pradhan and D. Avresky, eds., IEEE Computer Society Press), pp. 268-277, 1994.
19. Azevedo, M. M. de and Blough, D. M. "Software-based fault-tolerant clock synchronization for distributed UNIX environments," Technical Report ECE 94-03-01, Department of Electrical and Computer Engineering, University of California, Irvine, March 1994.