A simulation based study of Distributed Shared Memory System Issues

Ajay Mohindra   *Umakishore Ramachandran*

GIT–CC–92/48

October 1992

Abstract

Distributed shared memory (DSM) is a mechanism that provides shared memory abstraction in a physically distributed system. There are several issues, such as including the choice of protocol, the granularity of data transfer and coherence maintenance, and the hardware technology that have to be addressed in designing a DSM system. These issues together with the access pattern of the applications that are implemented on such an abstraction have a direct impact on the performance of a DSM system. A comprehensive workload model that captures the salient features of distributed programming paradigm is presented. This model allows specification of different application domains by tuning the parameters of the model. Using this model the system issues that affect the performance of a DSM system are studied via simulation for three algorithmic domains: transaction, iterative, and asynchronous. The key results are: Combining data transfer and synchronization is useful for the transaction domain; and relaxing strict-memory semantics is beneficial for the iterative and asynchronous domains.

Keywords: Distributed shared memory, Shared virtual memory, Workload models, Memory coherence protocols, Loosely-coupled systems.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280

1 Introduction

Distributed shared memory (DSM) is a popular paradigm for structuring distributed algorithms since it extends the well understood notion of shared memory to distributed systems. Several DSM systems have been designed and implemented [LH89, FP89, DLAR91, CBZ91], and there have been a few performance studies of DSM systems to evaluate the paging characteristics of specific coherence protocols [KL89]. However, there are several parameters that need to be considered for an efficient design of a DSM system. These include among others the choice of coherence protocol, the granularity of data transfer and coherence maintenance, and the hardware technology. Further, from the performance perspective the choice of these parameters are very much dependent on the characteristics of the applications that are implemented on top of DSM. Unfortunately, experience in developing applications that use the DSM paradigm is insufficient to draw hard conclusions on the choice of these parameters. To date, there have been no comprehensive studies that evaluate the performance implications of the different parameters from the point of view of applications that may use the DSM paradigm.

We have developed a versatile workload model that can be adapted to conform to different application domains. We define three different application domains: transaction processing, iterative algorithms, and asynchronous algorithms. For each of these domains we study via simulation the implications of the following system parameters on the performance of the DSM system: data transfer granularity, choice of protocol, the speed of the processors, and the speed of the network. The key findings of this study (Section 5) are: Combining data transfer and synchronization is useful for the transaction domain; and relaxing strict-memory semantics is beneficial for the iterative and asynchronous domains.

2 System Issues

2.1 Choice of Protocol

The choice of coherence protocol is related to the memory consistency model that is presented by the system. In our experiments (see Section 4.1), we consider three different protocols:

- *write-invalidate* – in this protocol the writer acquires exclusive ownership by invalidating all valid copies before performing the write [LH89].

- *lock-based* – in this protocol lock requests (exclusive and shared) result in the data associated with the lock being sent to the requester along with the granting of the lock. Upon release of a lock, the associated data is sent back (if modified) to the server. Reads or writes to shared data without explicit locking follow single-copy semantics that does not allow multiple-readers or writers. A weaker form of read allows multiple-readers to shared data (without locking) but does not guarantee consistency [RAK89].

- *buffered-update* – this protocol differentiates between two types of accesses: normal read/write and synchronization. Writes to shared data are buffered and consistency is enforced at synchronization points [CBZ91].

2.2 Data Transfer Granularity

Granularity of data transfer deals with the issue of the size of data that should be brought in by a node on a remote memory reference. Larger data transfer granularity may reduce the number of messages generated by the application, while at the same time allowing the application to exploit locality properties. On the other hand, this may also have the detrimental effect of increasing the amount of false sharing. The data transfer granularity may be at odds with the locking granularity that is used in the application implemented on top of DSM. For example, if the locking granularity is smaller than the data transfer granularity then it would limit the level of concurrency in a lock-based protocol since several program level locks may map to the same lock that governs the unit of data transfer assumed by the protocol; on the other hand, in the write-invalidate protocol false sharing would result since several data structures may be co-located in the same unit of data transfer assumed by the protocol.

2.3 Hardware Technology

There are two sources of overhead in a DSM system: the first is the communication overhead associated with the data transfer on the communication medium; and the second is the computational overhead associated with servicing remote memory requests. The choice of the communication medium (Ethernet, optical fiber, etc.) directly impacts the former, while the speed of the processor and any additional hardware support for DSM affects the latter. In our experiments we consider the impact of faster processors and faster communication medium on other system parameters.

3 The Workload Model

In this section we present a new workload model that is used as the basis to drive the simulator. This model is derived from the perceived characteristics of memory reference patterns in parallel and distributed programs. Archibald and Baer [AB86] have proposed a simple memory reference generator based on a probabilistic approach to evaluate cache coherence schemes in a shared memory multiprocessor. In their model, each processor generates a memory reference stream.

A memory reference (read or write) could either be to private or shared blocks; locality of references to shared blocks is modeled by increasing the probability for accesses to recently used shared blocks. The interaction between the memory reference streams of the different processors is simulated for different coherence protocols. A synthetic reference generator is used by Kessler and Livny [KL89] to evaluate distributed shared memory algorithms. The main difference from Archibald and Baer's model is that the memory reference stream of each processor is a sequence of shared and private phases. During a private phase the accesses are strictly to private memory, while both shared and private memory may be referenced during a shared phase. Each phase is characterized by length, placement, locality, read to write ratio, and type (private or shared).

Synchronization is an important aspect of any parallel program design, and the memory reference streams of processors executing a parallel program will consist of synchronization accesses and normal read/write accesses. Further, the interplay between distinct memory reference streams is dependent on the structure of the parallel program.

3.1 Structure

In our workload model, a parallel program is represented as a collection of tasks. The interrelationship between these tasks is captured by a task dependency graph, that suggests a partial execution order for the tasks that constitute the parallel program. A task is ready for execution when all tasks that precede it in the dependency graph have been completed. A work queue is maintained that contains the set of tasks that are ready for execution. Tasks are inserted into this queue honoring the dependencies in the task graph. A processor accesses the work queue to acquire a task to be executed next. When the work queue becomes empty, and all the tasks have terminated, the parallel program is said to have completed.

Each task is a memory reference stream of finite length (specified by a parameter) and is composed of a sequence of *compute* and *synchronization* phases. During a compute phase, the processor generates references (reads or writes) to private and shared data. A compute phase is characterized by the following parameters: the number of memory references, read to write ratio, probability for shared and private data accesses, and the degree of locality within the phase. The compute phase is similar to the shared phase as defined by Kessler, et al. A synchronization phase consists of read/write data accesses (both private and shared), with a percentage of the shared data accesses being done under the control of explicit synchronization.

The degree of locality within a phase defines the spatial locality for references within a page. In addition to this, the workload model allows designating distinct and disjoint regions of the shared address space to each task; and there is a parameter that governs the fraction of shared references of a task that are directed to its region as opposed to other tasks' regions. There is a parameter that controls the percentage of accesses to shared regions that are performed under the control of explicit synchronization (shared or exclusive). Figure 1 shows the composition of the the two phases within a task (the associated parameters are given in parentheses).

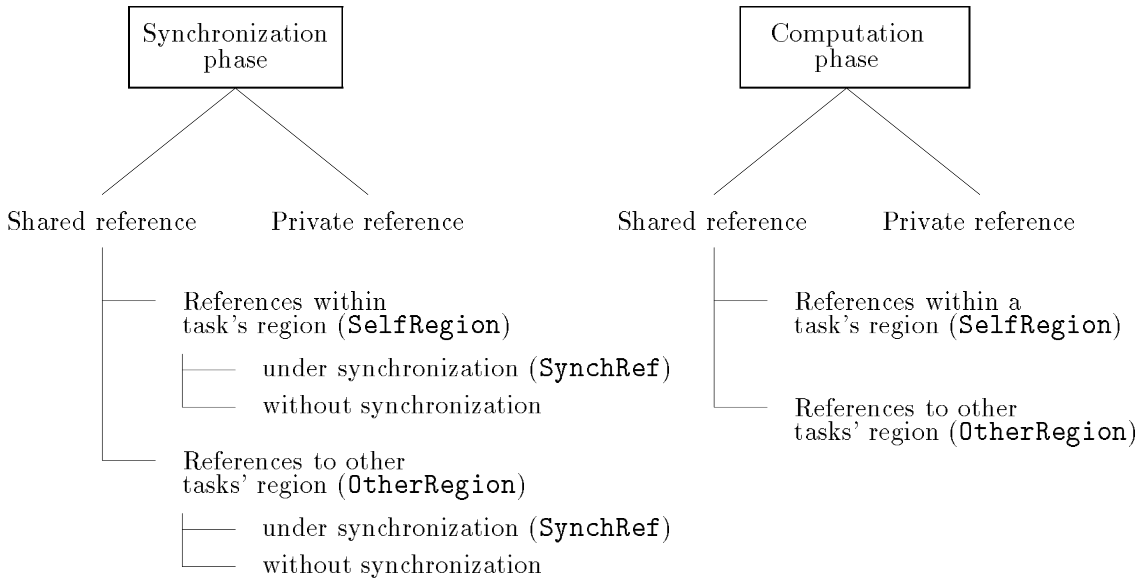

Figure 1: Reference parameters within a task

3.2 Domain specific models

By assigning proper values to the parameters shown in Figure 1, we can generate domain specific workload models. In our experiments, we used three different types of workload models representing three kinds of parallel programs.

1. **Transaction model**: This workload model reflects a transaction processing system wherein all references to shared data are made under the control of either shared or exclusive locks. Thus each task is composed of only synchronization phases, and all shared accesses are performed only under the control of explicit synchronization. (`SynchRef` = 1).

2. **Iterative Model**: Iterative algorithms such as linear equation solvers, have the characteristic that shared data is not modified except at well defined synchronization points (such as a barrier). Such a data access pattern would allow a task to access the shared

data without acquiring any locks for the purposes of reading. The iterative model captures this characteristic by allowing some percentage of the shared references (only reads) to be directed to other tasks' regions. (`OtherRegion` $\neq 0$, 0.05 in our experiments).

3. **Asynchronous Model**: In this model, tasks that comprise a computation do not synchronize with one another explicitly. In terms of the workload model this feature would translate to tasks reading and writing to shared memory without explicit synchronization. However an implementation of this model in a shared memory environment may involve the use of locks to govern access to mailboxes that may be used for asynchronous communication among the tasks. This workload model is similar to the data access patterns of asynchronous algorithms that rely on some other property such as convergence for correctness and termination [Bau78]. In terms of task parameters, some percentage of the shared references (both reads and writes) are directed to other tasks' regions. (`OtherRegion` $\neq 0$, 0.05 in our experiments).

4 Simulator

We have constructed a simulator to evaluate the design alternatives presented earlier. The simulator is written in CSIM [Sch86], a process oriented simulation language. The distributed system modeled by the simulator consists of a collection of nodes interconnected by a local area network. Each node in the network has a processor, a DSM coprocessor, and local memory. The local memory acts as a cache for the portion of the distributed shared memory currently residing at the node. The processor generates memory references as detailed in the earlier section. Remote memory references are serviced by the DSM coprocessor at each node in concert with other DSM coprocessors. In the simulator, each node is modeled as a set of three CSIM processes: a compute engine, a DSM server, and a media server. The interconnection network is modeled as a CSIM facility. The compute engine models a processor with associated local memory. Shared references which are not currently encached in the local memory are communicated to the DSM server by the compute engine. The DSM server simulates the appropriate coherence protocol. The media server models the communication subsystem of a node. It differentiates between two types of messages: CONTROL, and DATA. Each control message is 64 bytes long while the size of the data message is determined by the *page size* parameter used in the simulation. Currently the media server models the bandwidth characteristics of Ethernet and an optical fiber. It models the contention aspects of using a shared broadcast medium without modeling the collision and back-off aspects that are inherent in an Ethernet type of protocol. In addition to these three per node CSIM processes, a CSIM process serves as a centralized lock server.

In the simulator, the costs associated with performing the various chores in the implementation of a DSM system (such as state maintenance, communication processing overhead, page fault handling, and context switching) are taken from the Clouds distributed operating system [AMMR92].

Issues	Available alternatives
Data granularity (page size)	512, 1024, 2048, 4096, or 8192 bytes
Coherence protocol	write-invalidate, lock-based, or buffered-update
Communication medium	10 Mbps (Ethernet-like), or 1 Gbps (Fiber-like)
Processor Speeds	3 MIPS, 25 MIPS
Number of nodes	4, 8, 16

Table 1: List of studied alternatives

4.1 Parameters for the Simulation

We have designed a set of experiments to study the effects of the various design alternatives presented earlier. The approach we took is as follows: we used a set of compute nodes (3 MIPS CPU) connected by 10 Mbps Ethernet as the baseline system. We then designed our experiments to evaluate the effects of each issue on the performance of the overall system as compared to the baseline system. A list of issues that were studied is summarized in Table 1.

The experiments have been conducted for the three workload models described in the earlier section. An application is modeled as a 4-level deep task dependence graph, with 16 tasks at each level, yielding a total of 64 tasks. A task on level $i+1$ is not executed until all tasks at level i have been completed. Each task generates $100,000$ references. The lengths of the compute and the synchronization phases are specified as input parameters. The shared address space is 1MBytes divided into 128 logical segments of 8 Kbytes each. The logical segment is made up of physical pages; the page size is specified as an input parameter. Segment is a unit of locking assumed in the lock-based protocol while page is a unit of data transfer on remote memory request for all three protocols. For the purposes of this study it is assumed that the program level locks generated by the workload model map exactly to the segment level locks. This assumption essentially removes the effects (due to lock granularity and data transfer granularity mismatch) of false sharing for the invalidation-based protocol, and the effects of limited concurrency for the lock-based protocol.

In all our experiments, we fix the following parameters to be unchanged: 70% private data references, 80% reads, and 20% of shared references performed under explicit synchronization in the iterative and asynchronous workload models. The parameters that are varied in the experiments are summarized in Table 1. We use completion time as the metric for comparison.

All results reported in this paper are at 90% confidence level.

5 Experimental Results and Discussion

We present the results in two parts: First, we discuss the effects of granularity of data transfer, and choice of coherence protocol with respect to the three workload models. Second, we present the impact of the hardware technology on performance.

5.1 Transaction Model

One would expect that larger data granularity would reduce the number of messages in the system as fewer data requests are generated, and would increase spatial locality. However, larger data granularity also increases the potential for contention of shared data due to false sharing, thereby degrading system performance. In the transaction workload model (see Figure 2), we observe that the performance improves as the data granularity is increased for all three coherence protocols. False sharing is not an issue for this workload since all shared data references are performed under the control of a lock and since we assume lock granularity is a segment.

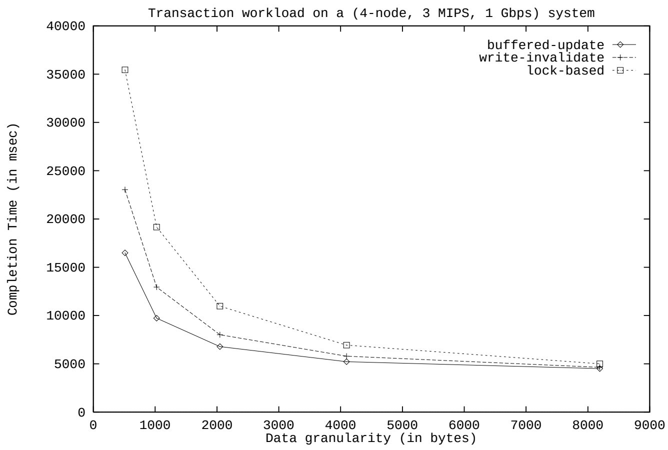

Figure 2: Transaction workload model's performance

The lock-based scheme is expected to incur a lesser number of messages on synchronized data accesses since it combines data transfer with synchronization. However, in this protocol the data pages associated with the lock are always shipped to the requestor along with the granting of the lock irrespective of whether the requestor has a valid copy or not. As can be seen in Figure 2, the lock-based protocol performs poorly at low data granularity compared to the other two. The reason is because at low data granularity more number of messages are required to bring in the

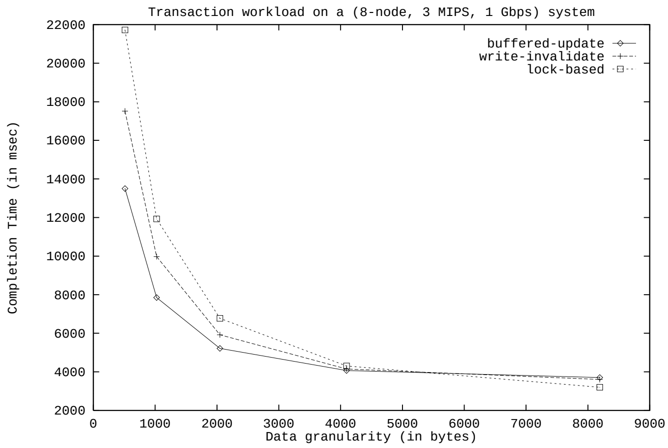

Figure 3: Transaction workload model's performance

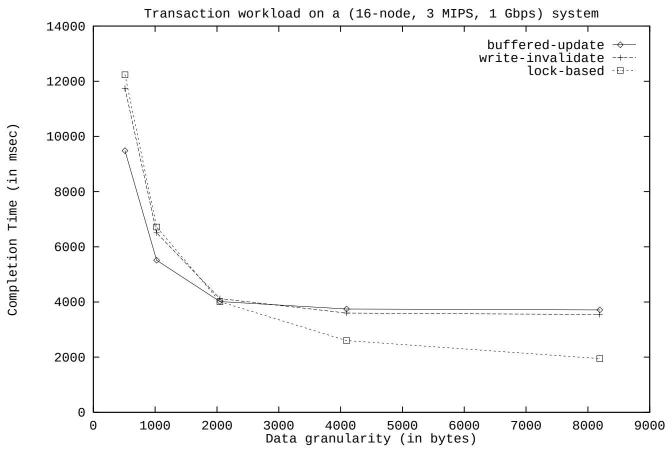

Figure 4: Transaction workload model's performance

entire segment associated with the lock. The write-invalidate and buffered-update schemes may not have to incur this message overhead if the data is valid at the requestor. However at higher data granularity the lock-based scheme performs better since the number of messages per lock request reduces significantly. Overall the buffered-update scheme performs better than either the write-invalidate or lock-based scheme (see Figure 2), although at large data granularity, the difference between the write-invalidate and buffered-update scheme is statistically insignificant. In the buffered-update scheme, only the updates are sent to the server at synchronization points, and further the protocol does not incur the overhead of invalidation messages. It is interesting to note for larger systems (8 and 16 nodes) the lock-based scheme performs much better than the other two schemes for large data granularity (see Figures 3 and 4). For write-invalidation scheme, the probability of the data associated with a lock being valid decreases due to the increased concurrent activity over the same number of shared segments. Similarly, for buffered-update scheme the updates are sent to the current set of potential readers and all of them may not actually use it in the future. On the other hand, the lock-based protocol incurs exactly the minimum number of messages required to get the lock and data. As we increase the number of nodes in the system, the number of messages becomes an important factor (due to contention for the communication medium) in determining the system performance.

5.2 Iterative Model

Recall that the iterative workload model (see Section 3.2) allows a task to access shared data for reading without explicitly acquiring read-locks. For this model, increasing data transfer granularity improves system performance for the lock-based and buffered-update schemes (see Figure 5). However, for the write-invalidate scheme, the performance benefit due to the reduced number of messages (at larger data granularity) is offset by an increase in false sharing, thus resulting in system performance degradation. Since read-shared copies are invalidated upon a write, the cost of re-reading a new valid copy increases with increasing data granularity for a given sharing pattern. The problem becomes even more acute when more nodes are added to the system, as now it is more likely that read-shared data pages may become invalid (see Figure 6). Since false sharing is not an issue with either the lock-based or buffered-update protocols, we do not see a similar performance degradation with either of these protocols.

Since both the lock-based and buffered-update schemes allow the copies of shared data to remain inconsistent between synchronization points, these two are expected to perform better than write-invalidate scheme for the iterative workload model. Figures 5 and 6 confirm this hypothesis. However it is surprising that buffered-update scheme does not do as well as lock-based scheme. In the buffered-update scheme, updates for all modified pages are sent at the end of each synchronization epoch. This set of pages could potentially include ones that are

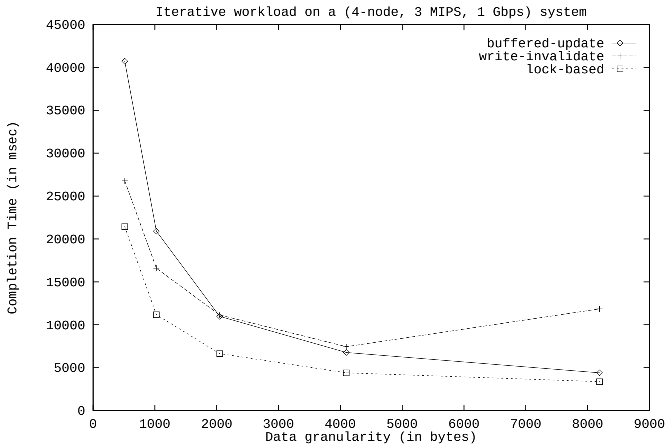

Figure 5: Iterative workload model's performance

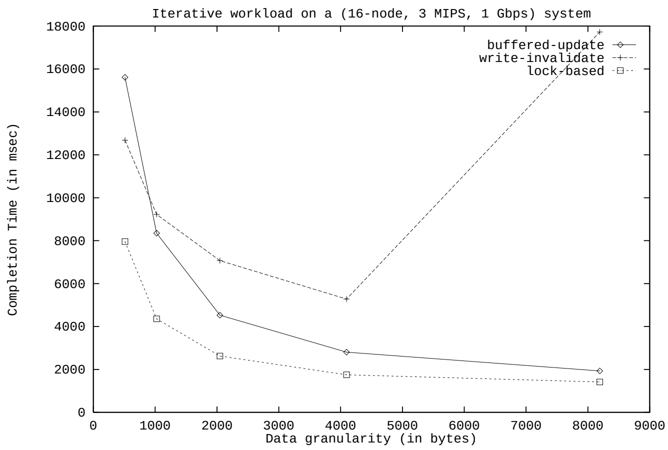

Figure 6: Iterative workload model's performance

unrelated to this particular epoch. As a result this scheme could incur more overhead than entirely called for in the iterative workload model. The lock-based scheme by associating locks with segments does not have to incur this unnecessary overhead. This effect is more apparent at low data granularities (small page sizes). In fact, as can be seen even write-invalidate scheme performs better than buffered-update scheme at sufficiently small data granularity since the need for unnecessary updates in the latter over-shadows the ill-effect of false-sharing in the former. At higher data granularities the distinction between lock-based and buffered-update schemes is lesser.

5.3 Asynchronous Model

In this model, unsynchronized write-sharing of data is allowed (see Section 3.2). Further the domain of write-shared data is the entire shared data space. Thus the model itself has a high built-in overhead (as compared to the iterative model) for both the write-invalidate or buffered-update style protocols. In the former, invalidations may have to be sent to all the nodes while in the latter updates may have to be sent to all the nodes. This is evident by comparing absolute completion times for the same amount of total work (in terms of number of memory references) for the two workload models (see Figures 5 and 8). As can be seen from Figure 8 increasing the data granularity helps both the protocols. The positive effect of reducing the number of messages at larger data granularities seems to dominate the negative effect of false-sharing for the write-invalidate protocol. The lock-based protocol (owing to its assumption that computations obey a synchronization model) is basically incompatible with this asynchronous workload model. Owing to the protocol allowing exactly one-copy of a segment (regardless of the data granularity) for such asynchronous accesses the lock-based protocol performs consistently worse than the other two for all data granularity (see Figure 7). However due to lesser number of messages at larger granularities the performance of lock-based protocol approaches that of the other two.

5.4 Hardware Technology

We conducted several experiments to determine the effects of new technology on the overall performance. When the processors in the baseline system are replaced with faster processors (see Figure 9), the overall system performance improves, although the percentage improvement due to reduce computation times is not uniform across all data transfer sizes. The non-uniform improvement across the range of data granularity can be explained as follows: For low data granularity, more number of data requests are generated, thereby increasing the computation requirements associated with page-fault handling and DSM related state maintenance; as a

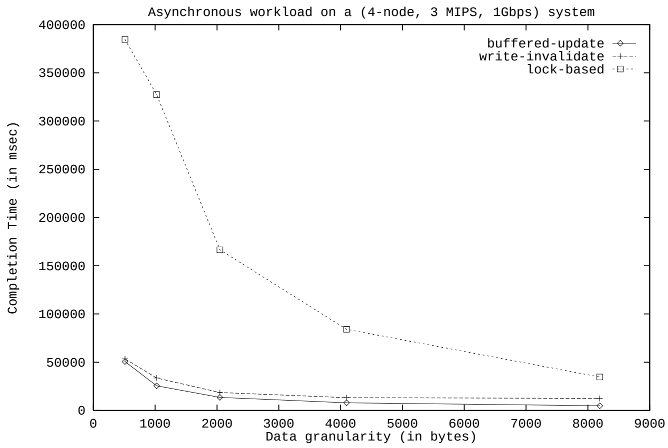

Figure 7: Asynchronous workload model's performance

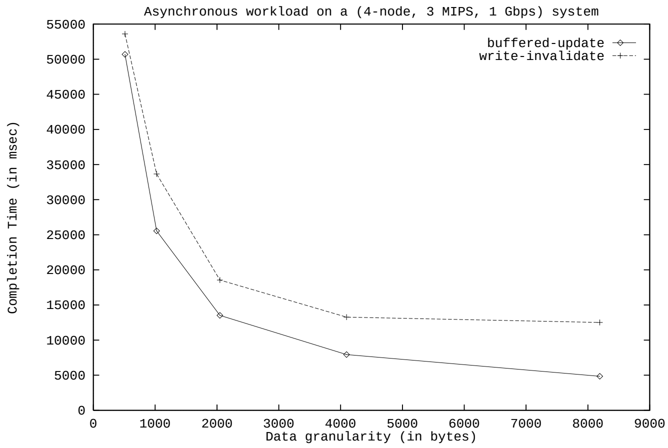

Figure 8: Asynchronous workload model's performance

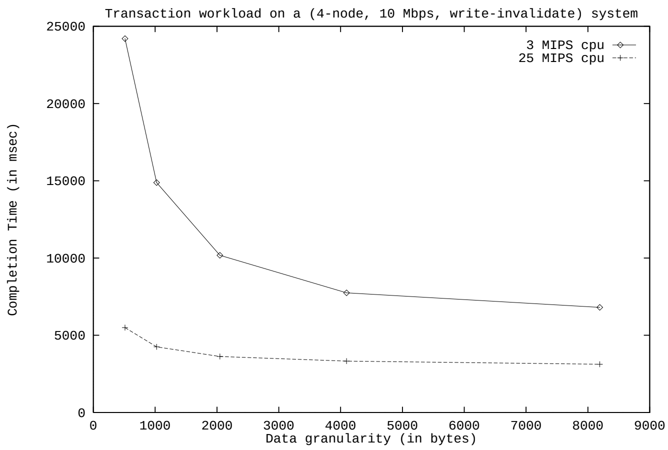

Figure 9: Effect of processor speed on performance

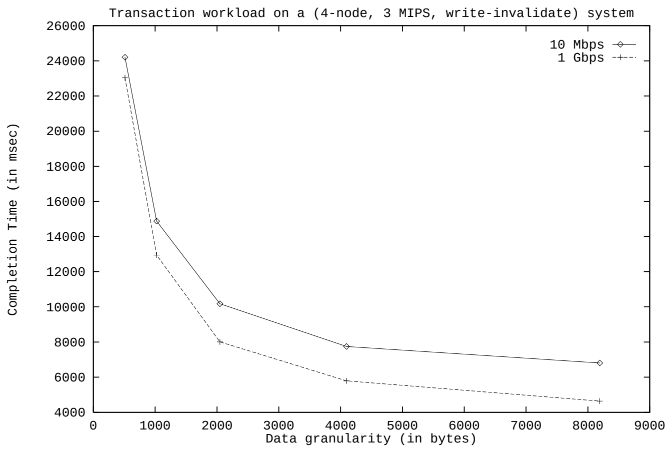

Figure 10: Effect of communication speed on performance

result, the processor speed has a significant impact on the performance than for large data granularity. Similar performance improvement is observed when the communication medium in the baseline system is replaced with a faster communication medium (see Figure 10). The reason for the improvement is reduced transmission times. The impact of communication speed on the performance becomes more significant as the data granularity is increased because at low data granularity the access to the medium is the primary source bottleneck (due to large number of messages).

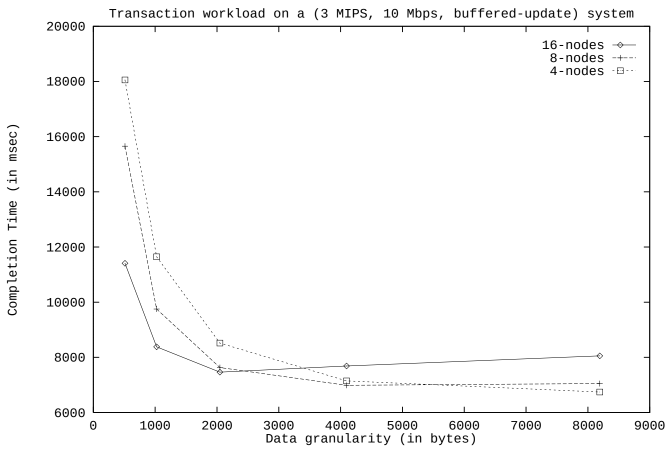

Figure 11: System scalability with 3 MIPS CPU and 10 Mbps communication medium

5.4.1 Effects of technology factors on system scalability

We define scalability of a distributed system as the effect of increased number of nodes in the system on the performance of the problem being solved, e.g., if a problem of size $\mathcal{N}$ is solved in time t_1 by $\mathcal{P}$ processors then it should be solved by $k\mathcal{P}$ ($k > 1$) processors in time t_2 where $t_2 \leq t_1$. Ideally, we would like $t_1/t_2 = k$. In this case, the problem is said to achieve linear speedup.

To study the effects of technology factors on system scalability, we examined a distributed system using the transaction model and the buffered-update scheme. Figures 11, 13, 12, and 14 show the results for four possible system configurations: 3 MIPS CPU with 10 Mbps and 1 Gbps medium, and 25 MIPS CPU with 10 Mbps and 1 Gbps medium.

Figure 11 shows the system performance as a function of data transfer granularity for 4-, 8-, and 16-node configurations. Although for a given number of processors the system performance improves as the data transfer granularity is increased, the performance degrades as the number of processors is increased for a given data granularity. This is because the increase in computational power through the additional processors is not matched by the available communication bandwidth. The scalability problem becomes even more acute when faster processors are used in the system without changing the communication medium (see Figure 12). Similar behavior is observed for the iterative and the asynchronous models also.

The degradation in system scalability can be eliminated by using a faster communication medium (see Figures 13 and 14). Note however that for 16 processors there is very little improvement in performance with increased data granularity. In the transaction workload model the processors compete for the same fixed number of locks. There are two sources of

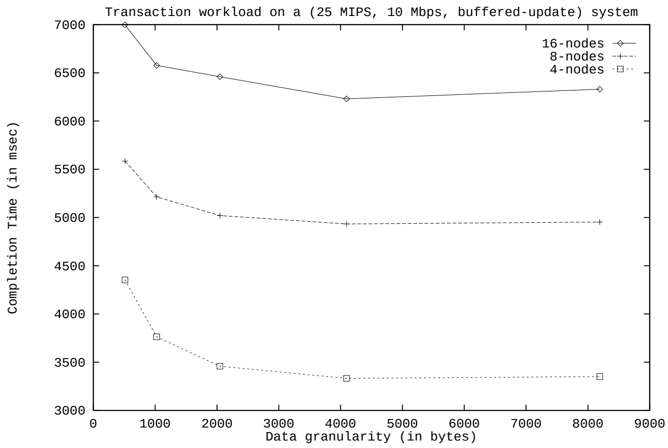

Figure 12: System scalability with 25 MIPS CPU and 10 Mbps communication medium

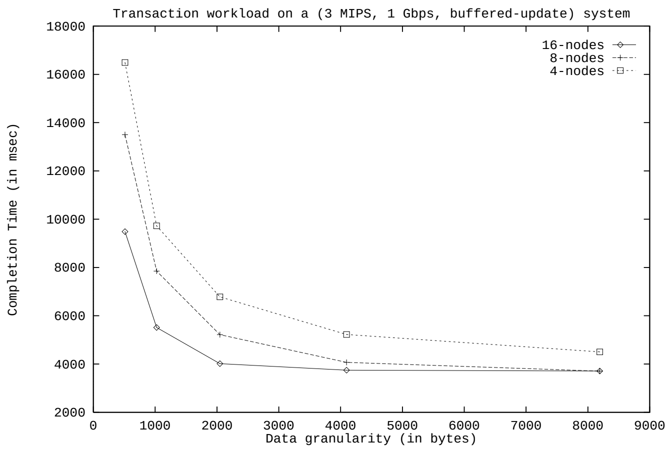

Figure 13: System scalability with 3 MIPS CPU and 1 Gbps communication medium

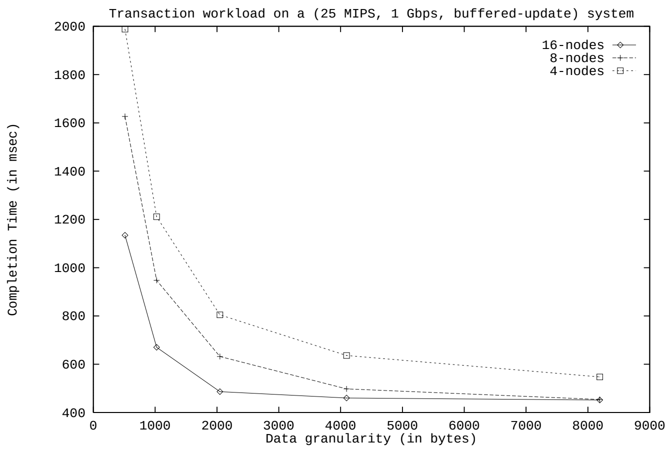

Figure 14: System scalability with 25 MIPS CPU and 1 Gbps communication medium

overhead in this model: one corresponding to data transfer and the other corresponding to the waiting time for lock service. The data transfer overhead is dependent on the number of messages and the size of each message. Since for the configurations studied the processing overhead per message dominates the actual transmission time on the wire, the number of messages is the more critical factor in determining the data transfer overhead. At low data granularity the number of messages is higher but the waiting time is higher; while at high data granularity the number of messages is smaller but due to the increased service time for each lock request (owing to the larger data granularity) the waiting time for locks is higher. Thus the two sources of overhead balance each other out resulting in no net gain in performance for large data granularities when the size of the system is scaled up.

5.5 Effects of a DSM Coprocessor

In our experiments so far we assumed that each node has a host processor and a DSM coprocessor. We conducted a set of experiments to study the effect of sharing the host processor to perform DSM related activity. Figure 15 compares the results for two such system configurations; one with a DSM coprocessor and one without a DSM coprocessor. As shown in Figure 15, the system with a DSM coprocessor shows much better performance than than one without. The improvement is more significant for small data transfer sizes than for large data sizes. The reason is that for small data transfer sizes, more number of DSM requests are generated in the system, thereby requiring more DSM related work to be performed at each node. Under such conditions, existence of a DSM coprocessor improves the system performance. Another result that is observed is the performance improvement due to the presence of a DSM coprocessor becomes more significant as the number of processors is increased. For example, for a data

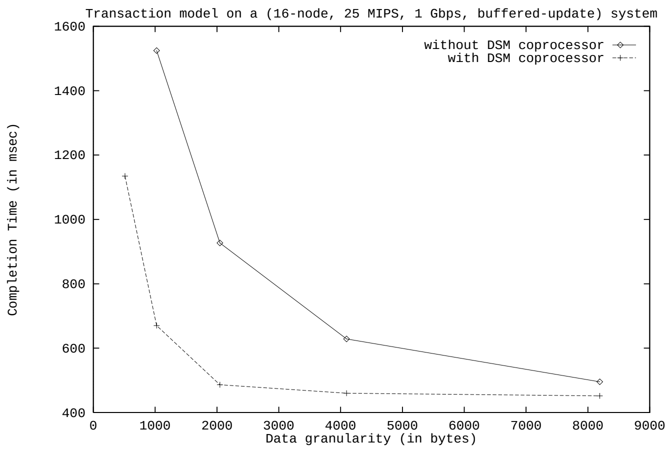

Figure 15: Effect of using a DSM coprocessor on the performance

transfer size of 2 Kbytes, 13% improvement was observed in a 4-node configuration, 25% in a 8-node configuration, and 47% in a 16-node configuration. This behavior is consistent with the observation that an increase in the number of nodes is accompanied by an increase in the amount of DSM related activity in the system.

6 Conclusions

In the paper, we present a versatile workload model that captures the salient features of parallel and distributed applications. The workload model is used to study (via simulation) the system issues that affect the design and performance of distributed shared memory systems. The key findings of the study are as follows. Combining data transfer and synchronization makes sense for the transaction workload, especially for large system configurations. The weaker memory semantics of the lock-based protocol matches the requirements of the iterative workload, while the buffered-update protocol performs well for the asynchronous workload.

References

[AB86] J. Archibald and Jean-Loup Baer. Cache coherence protocols: Evaluation using a multiprocessor s imulation model. *ACM Transactions on Computer Systems*, 4(4):273 – 298, Nov 1986.

[AMMR92] R. Ananthanarayanan, Sathis Menon, Ajay Mohindra, and Umakishore Ramachandran. Experiences in integrating distributed shared memory with virtual memory management. *Operating Systems Review*, 26(3):4–26, 1992.

[Bau78] G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM*, 25(2):226–244, April 1978.

[CBZ91] J. B. Carter, J. K. Bennett, and W. Zwaenepoel. Implementation and performance of Munin. In *The 13th ACM Symposium on Operating Systems Principles*, pages 152–164, October 1991.

[DLAR91] Partha Dasgupta, Richard LeBlanc, Mustaque Ahamad, and Umakishore Ramachandran. The Clouds distributed operating system. *IEEE Computer*, April 1991.

[FP89] Brett D. Fleisch and Gerald J. Popek. Mirage: A coherent distributed shared memory design. *Operating Systems Review*, 23(5):211–223, Dec 1989.

[KL89] R. E. Kessler and Miron Livny. An analysis of distributed shared memory algorithms. In *9th Intl Conference on Distributed Computing Systems*, pages 498–505, 1989.

[LH89] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321 – 359, Nov 1989.

[RAK89] Umakishore Ramachandran, Mustaque Ahamad, and M. Yousef A. Khalidi. Coherence of distributed shared memory: Unifying synchronization and data transfer. In *18th International Conference on Parallel Processing*, pages 160–169, Aug 1989.

[Sch86] H. D. Schwetman. CSIM: A C-based, process-oriented simulation language. In *Proceedings of the 1986 Winter Simulation Conference*, pages 387–396, December 1986.