ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Review on Cache-Coherent Algorithms with Multithreaded Computer Architectures

T.Karthik[1], M.Krishna Sudha[2]

M.Phil Research Scholar, Sri Vasavi College, Erode, India [1]

Head, Department of Information Technology, Sri Vasavi College, Erode, India [2]

Abstract: Cache-coherent multithreaded computer architectures has been envisioned as the next generation architecture of IT Enterprise. In contract to traditional solutions, where the IT services are under proper physical, logical and personnel controls, Cloud Computing moves the application software and database to the large data centers, where the management of the data and services may not be fully trustworthy. This unique attribute, however, poses many new security challenges which have not been well understood. In this article, we focus on cloud data storage security, which has always been an important aspect of quality services. To ensure the correctness of users' data in the cloud, we propose effective and flexible distributed schemes with two salient features, opposing to its predecessors. By utilizing the homomorphism token with distributed verification of erasure-coded data, our scheme achieves the integration of storage correctness insurance and data error localization, i.e., the identification of misbehaving server(s). Unlike most prior works, the new scheme further supports secure and efficient dynamic operations on data blocks, including: data update, delete and append. Extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attacks, and even server cache-coherent attacks.

Keywords: Cache, Multithreading, Homomorphism, Resilient, Simulation.

I. INTRODUCTION

System availability is crucial for the multi-threaded (including multiprocessor) systems that run critical infrastructure. Unless architectural steps are taken, availability will decrease over time, as implementations use larger numbers of increasingly unreliable components in search of higher performance. Both the industry and the academics predict that transient and permanent faults will lead to increasing hardware error rates. Backward error recovery is a cost-effective mechanism to tolerate runtime hardware errors, but it can only recover from errors that are detected in a timely fashion. Traditionally, most systems employ localized error detection mechanisms such as parity bits on cache lines and memory buses to detect errors. Although such specialized mechanisms detect the errors that they target, they do not comprehensively detect whether the end-to-end behavior of the system is correct.

Our goal is end-to-end error detection for multithreaded memory systems, which would subsume localized mechanisms and provide comprehensive error detection. Memory systems are complicated concurrent systems that include caches, memories, coherence controllers, interconnection network, and all of the other glue that enables multiple processor cores or hardware thread contexts to communicate. As more of the memory system becomes integrated on chip, including cache and memory controllers and logic for glueless multi chip multiprocessing, this logic becomes just as susceptible to hardware errors as the processor core logic. In this paper, we focus on the runtime detection of transient and permanent hardware errors in the memory system.

We do not consider the orthogonal problem of detecting errors unrelated to the memory system in processor cores, because good solutions to this problem already exist (for example, redundant multithreading and diva).our previous work achieved end-to-end error detection for a very restricted class of multithreaded memory systems. In that work, we designed an all-hardware scheme for the dynamic verification of sequential consistency (dvsc), which is the most restrictive consistency model. Since the end-to-end correctness of a multithreaded memory system is defined by its memory consistency model, dvsc comprehensively detects errors in systems that implement sequential consistency (sc). However, dvsc's applications are limited, because sc is not frequently implemented.

II. LITERATURE REVIEW

Shared Memory Consistency Models : This paper presents an overview of the field of memory consistency model of a system affects the performance, programmability, and portability. We described memory consistency models in a way that most computer professionals would understand. This is important if the performance-enhancing features being incorporated by system designers are to be correctly and widely used by programmers. Our focus is consistency models proposed for hardware-based shared memory systems. Most of these models emphasize the system optimizations they support, and we retain this system-centric emphasis. We also describe an alternative, programmer-centric view of relaxed consistency models that describes them in terms of program behavior, not system optimizations

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Dynamic verification of sequential consistency : In this paper, we develop the first feasibly implemental scheme for end-to-end dynamic verification of multithreaded memory systems. For multithreaded (including multiprocessor) memory systems, end-to-end correctness is defined by its memory consistency model. One such consistency model is sequential consistency (SC), which specifies that all loads and stores appear to execute in a total order that respects program order for each thread. Our design, DVSC-Indirect, performs dynamic verification of SC (DVSC) by dynamically verifying a set of sub-invariants that, when taken together, have been proven equivalent to SC. We evaluate DVSC-Indirect with full-system simulation and commercial workloads. Our results for multiprocessor systems with both directory and snooping cache coherence show that DVSC-Indirect detects all injected errors that affect system correctness (i.e., SC). We show that it uses only a small amount more bandwidth (less than 25%) than an unprotected system and thus can achieve comparable performance when provided with only modest additional link bandwidth

Algorithms For Dynamic Software Cache Coherence: In this paper, we investigate a class of cache coherence strategies for shared data at the program-level, referred to as Shared Regions (SR), is used to manage caches dynamically through software. The practical value of these strategies is measured by their performance relative to existing hardware coherence protocols, and the complexity of the SR programming interface. We present detailed quantitative results highlighting the performance of a wide array of SR coherence algorithms, including some novel algorithms introduced in this paper that use direct cache-to-cache data transfers via software to improve performance. These algorithms are studied using execution-driven simulation and compared to a representative hardware strategy for a suite of parallel applications. We study the issue of programming complexity by analyzing the process of inserting Shared Regions program annotations into these applications.

Memory Ordering: A Value-Based Approach: Conventional out-of-order processors employ a multi-ported, fully-associative load queue to guarantee correct memory reference order both within a single thread of execution and across threads in a multiprocessor system. As improvements in process technology and pipelining lead to higher clock frequencies, scaling this complex structure to accommodate a larger number of in-flight loads becomes difficult if not impossible. Furthermore, each access to this complex structure consumes excessive amounts of energy. In this paper, we solve the associative load queue scalability problem by completely eliminating the associative load queue. Instead, data dependences and memory consistency constraints are enforced by simply reexecuting load instructions in program order prior to retirement. Using heuristics to filter the set of loads that must be re-executed, we show that our replay-based mechanism enables a simple, scalable, and energy-efficient FIFO load queue design with no associative lookup functionality, while sacrificing only a negligible amount of performance and cache bandwidth.

Verification techniques for cache coherence protocols : In this article we present a comprehensive survey of various approaches for the verification of cache coherence protocols based on state enumeration, (symbolic model checking, and symbolic state models. Since these techniques search the state space of the protocol exhaustively, the amount of memory required to manipulate that state information and the verification time grow very fast with the number of processors and the complexity of the protocol mechanisms. To be successful for systems of arbitrary complexity, a verification technique must solve this so-called state space explosion problem. The emphasis of our discussion is on the underlying theory in each method of handling the state space exposion problem, and formulating and checking the safety properties (e.g., data consistency) and the liveness properties (absence of deadlock and live lock). We compare the efficiency and discuss the limitations of each technique in terms of memory and computation time. Also, we discuss issues of generality, applicability, automaticity, and amenity for existing tools in each class of methods. No method is truly superior because each method has its own strengths and weaknesses. Finally, refinements that can further reduce the verification time and/or the memory requirement are also discussed.

Improving the throughput of synchronization by insertion of delays : Efficiency of synchronization mechanisms can limit the parallel performance of many shared-memory applications. In addition, the ever increasing performance gap between processor and interprocessor communication may further compromise the scalability of these primitives. Ideally, synchronization primitives should provide high performance under both high and low contention without requiring substantial programmer effort and software support. QOLR has been shown to offer substantial speedups and to outperform other synchronization primitives consistently, but at the cost of software support and protocol complexity. This paper proposes the use of speculation and delays to implement a purely hardware-based queueing mechanism called Implicit QOLB. Making use of the pervasiveness of the Load-Linked/Store-Conditional primitives, we present a series of hardware mechanisms to optimize performance for sharing patterns exhibited by locks and associated data. The mechanisms do not require any change to existing software or instruction sets. IQOLB sits alongside the cache-coherence protocol and guides the decisions the protocol makes with respect to lock (and associated data) transfers. Preliminary evaluations indicate that IQOLB may perform as well as, if not better than, QOLB without the additional software and protocol complexity.

Automatic Verification of Cache Coherence: State-based, formal methods have been successfully applied to

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

the automatic verification of cache coherence in sequentially consistent systems. However, coherence in shared memory multiprocessors under a relaxed memory model is much more complex to verify automatically. With relaxed memory models, incoming invalidations and outgoing updates can be delayed in each cache while processors are allowed to race ahead. This buffering of memory accesses considerably increases the amount of state in each cache and the complexity of protocol interactions. Moreover, because caches can hold inconsistent copies of the same data for long periods of time, coherence cannot be verified by simply checking that cached copies are identical at all times. This paper makes two major contributions. First, we demonstrate how to model and verify cache coherence under a relaxed memory model in the context of state-based verification methods. Frameworks for modeling the hardware and for generating correct memory access sequences driving the hardware model are developed. We also show correctness properties which must be verified on the hardware model. Second, we demonstrate a successful application of a state-based verification tool called SSM for the verification of the delayed protocol, an aggressive protocol for relaxed memory models. SSM is based on an abstraction technique preserving the properties to verify. We show that with classical, explicit approaches the verification of cache coherence is realistically unfeasible because of the state space explosion problem, whereas SSM is able to verify protocols both at both behavioral and message-passing levels.

III. PROPOSED SYSTEM

Security threats faced by cache-coherent data storage can come from two different sources. On the one hand, a CSP can be self-interested, untrusted and possibly malicious. Not only does it desire to move data that has not been or is rarely accessed to a lower tier of storage than agreed for monetary reasons, but it may also attempt to hide a data loss incident due to management errors, Byzantine failures and so on. On the other hand, there may also exist an economically motivated adversary, who has the capability to compromise a number of cache-coherent data storage servers in different time intervals and subsequently is able to modify or delete users' data while remaining undetected by CSP for a certain period..

Specifically, we consider two types of adversary with different levels of capability in this system. Weak adversary is interested in corrupting the user's data files stored on individual servers. Once a server is comprised, An adversary can pollute the original data files by modifying or introducing its own fraudulent data files by modifying or introducing its own fraudulent data to prevent the original data from being retrieved by the user.Strong Adversaary is the worst case scenario, in which we assume that the adversary can compromise all the storage servers so that he can intentionally modify the data files as long as they are internally consistent.In fact, this is equivalent to the case where all servers are colluding together to hide a data loss or corruption incident.

IV. ALGORITHM REVIEW

In computing, cache algorithms (also frequently called replacement algorithms or replacement policies) are optimizinginstructions – or algorithms – that a computer program or a hardware-maintained structure can follow, in order to manage a cache of information stored on the computer. When the cache is full, the algorithm must choose which items to discard to make room for the new ones.

The average memory reference time is

$$T = m * T_m + T_h + E$$

where

T = average memory reference time

m = miss ratio = 1 - (hit ratio)

T_m = time to make a main memory access when there is a miss (or, with multi-level cache, average memory reference time for the next-lower cache)

T_h= the latency: the time to reference the cache when there is a hit

E = various secondary effects, such as queuing effects in multiprocessor systems

There are two primary figures of merit of a cache: The latency, and the hit rate. There are also a number of secondary factors affecting cache performance.The "hit ratio" of a cache describes how often a searched-for item is actually found in the cache. More efficient replacement policies keep track of more usage information in order to improve the hit rate (for a given cache size).The "latency" of a cache describes how long after requesting a desired item the cache can return that item (when there is a hit). Faster replacement strategies typically keep track of less usage information—or, in the case of direct-mapped cache, no information—to reduce the amount of time required to update that information. Each replacement strategy is a compromise between hit rate and latency. Measurements of "the hit ratio" are typically performed on benchmark applications. The actual hit ratio varies widely from one application to another. In particular, video and audio streaming applications often have a hit ratio close to zero, because each bit of data in the stream is read once for the first time (a compulsory miss), used, and then never read or written again. Even worse, many cache algorithms (in particular, LRU) allow this streaming data to fill the cache, pushing out of the cache information that will be used again soon (cache pollution).

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

Least Recently Used (LRU) : Discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards the least recently used item. General implementations of this technique require keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes. LRU is actually a family of caching algorithms.

```
 public void CleanUp( DateTime now )
{
   if( _current != _oldest )
      lock( this )
      {
         //calculate how many items should be removed
         DateTime maxAge = now.Subtract( _maxAge );
         DateTime minAge = now.Subtract( _minAge );
         int itemsToRemove = _owner._curCount -
_owner._capacity;
         AgeBag bag = _bags[_oldest % _size];
         while( _current != _oldest && (_current-
_oldest>_size - 5
               || bag.startTime < maxAge || (itemsToRemove
> 0
               && bag.stopTime > minAge)) )
         {
            // cache is still too big / old so remove oldest bag
            Node node = bag.first;
            bag.first = null;
            while( node != null )
            {
               Node next = node.next;
               node.next = null;
               if( node.Value != null && node.ageBag !=
null )
                  if( node.ageBag == bag )
                  {
                     // item has not been touched since bag
was
                     // closed, so remove it from LifespanMgr
                     ++itemsToRemove;
                     node.ageBag = null;
                     Interlocked.Decrement( ref
_owner._curCount );
                  }
                  else
                  {
                     // item has been touched and should
                     // be moved to correct age bag now
                     node.next = node.ageBag.first;
                     node.ageBag.first = node;
                  }
               node = next;
            }
            // increment oldest bag
            bag = _bags[(++_oldest) % _size];
         }
         OpenCurrentBag( now, ++_current );
         CheckIndexValid();
      }
}
```

Pseudo-LRU (PLRU): For CPU caches with large associativity (generally >4 ways), the implementation cost of LRU becomes prohibitive. In many CPU caches, a scheme that almost always discards one of the least recently used items is sufficient. So many CPU designers choose a PLRU algorithm which only needs one bit per cache item to work. PLRU typically has a slightly worse miss ratio, has a slightly better latency, and uses slightly less power than LRUWhich memory locations can be cached by which cache locations

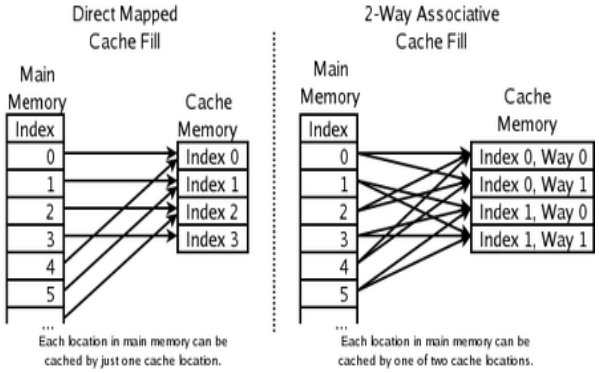

Fig 1: cache Coherent simulations method

Random Replacement (RR) : Randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history. For its simplicity, it has been used in ARM processors.It admits efficient stochastic simulation.

V. CONCLUSION

This paper presents a framework that can dynamically verify a wide range of consistency models and comprehensively detect memory system errors. Our verification framework is modular, because it checks three independent invariants that together are sufficient to guarantee memory consistency. The modular design makes it possible to replace any of our checking mechanisms with a different scheme to adapt to a specific system's design. For example, the coherence checker adapted from DVSC can be replaced by the design proposed DVMC is also not limited to the conventional multiprocessor systems described in this paper but could be used with other shared memory based architectures. The simplicity of the proposed mechanisms suggests that they can be implemented with small modifications to existing multi-threaded systems. Simulation of a DVMC implementation shows some decrease in performance, but we expect the negative impact to be outweighed by the benefit of improved safety and availability.

REFERENCES

[1] S.V. Adve and K. Gharachorloo, "Shared Memory Consistency Models: A Tutorial," Computer, vol. 29, no. 12, pp. 66-76, Dec. 1996.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 6, June 2015

[2] A.R. Alameldeen, M.M. Martin, C.J. Mauer, K.E. Moore, M. Xu, M.D. Hill, D.A. Wood, and D.J. Sorin, "Simulating a $2M Commercial Server on a $2K PC," Computer, vol. 36, no. 2, pp. 50-57, Feb. 2003.

[3] T.M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," Proc. 32nd Ann. IEEE/ACM Int'l Symp.Microarchitecture (MICRO '99), pp. 196-207, Nov. 1999

[4] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation,"Proc. ACM Sigmetrics '98, pp. 151-160, June 1998.

[5] H.W. Cain and M.H. Lipasti, "Verifying Sequential Consistency Using Vector Clocks," Proc. 14th ACM Symp. Parallel Algorithms and Architectures (SPAA '02), Aug. 2002.

[6] H.W. Cain and M.H. Lipasti, "Memory Ordering: A Value-Based pproach," Proc. 31st Ann. Int'l Symp. Computer Architecture(ISCA '04), June 2004.

[7] J.F. Cantin, M.H. Lipasti, and J.E. Smith, "Dynamic Verification of Cache Coherence Protocols," Proc. Second Workshop Memory Performance Issues (WPI '01), June 2001.

[8] S. Chatterjee, C. Weaver, and T. Austin, "Efficient Checker Processor Design," Proc. 33rd Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO '00), pp. 87-97, Dec. 2000.

[9] K. Gharachorloo, A. Gupta, and J. Hennessy, "Two Techniques to Enhance the Performance of Memory Consistency Models," Proc. Int'l Conf. Parallel Processing (ICPP '91), vol. I, pp. 355-364, Aug. 1991.

[10] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory Consistency and Event Ordering in Scalable Shared-Memory," Proc. 17th Ann. Int'l Symp. Computer Architecture (ISCA '90), pp. 15-26, May 1990.

BIOGRAPHIES

T.Karthik obtained his M.Sc software engineering. He is currently doing his M.Phil Research Programme from Sri Vasavi College, erode. His research interest include data mining, Text mining and soft computing

M.Krishna sudha is Head, Department of Information Technology, Sri Vasavi College, Erode, Tamilnadu, India with more than 10 years of teaching eperience.She Obtained her MCA Degree from Government Arts college, Coimbatore, Tamilnadu, India. She completed her Mphil Research Programme From Bharathiayar university, Coimbatore and persuing her PhD Program . She published more than 10 papers in reputed international journals and produced more than 15 mphil scholars. Her research Interest Include Networking, Cloud Computing, Soft Computing and web services