Analyzing Cache Coherence Protocols for Server Consolidation

Antonio García-Guirado, Ricardo Fernández-Pascual, José M. García
Dept. de Ingeniería y Tecnología de Computadores
Universidad de Murcia
Murcia, Spain
{toni, r.fernandez, jmgarcia}@ditec.um.es

***Abstract*—Server consolidation is commonly used today to make the most out of all the cores of a chip multiprocessor by running several virtual machines (VMs) on it. Cache coherence protocols can be adapted to take advantage of such an scenario. In this line, Virtual Hierarchies (VHs) use two levels of cache coherence in a consolidated server. They isolate the coherence actions of each VM and improve performance by maximizing the number of memory accesses serviced by caches within the VM. In this paper we show how hierarchical protocols with no single ordering point for the requests, such as VHs in the form currently proposed, are prone to deadlocks. Besides, when memory deduplication is used, VHs cannot take advantage of memory deduplication at the cache level, both because deduplicated data is reduplicated in cache, and because accesses to deduplicated data often require the access to the cache tiles used by a different VM by means of broadcast. We analyze all these problems and we propose solutions for them, showing the actual performance of these protocols, and giving some insights for the future development of coherence protocols optimized for server consolidation.**

I. Introduction

Nowadays, the number of processor cores in mainstream computers is increasing. In particular, chip multiprocessors (CMPs) are commonly used in servers. To take advantage of these circumstances, server consolidation is commonly used to run several services in a single server, usually by means of running several virtual machines (VMs) on a single physical machine.

On the other hand, current cache coherence protocols do not scale well with the number of cores. Directory-based and token-based protocols [1] are the most promising solutions to keep the cache coherence in such machines, but these protocols show a number of problems as the number of processors grows. Traditional directory protocols add an extra level of indirection when solving a cache miss. The average distance that the messages must travel due to this indirection grows with the number of processors in the CMP. On the other hand, token protocols usually rely on broadcast to solve requests, which leads to contention in the interconnection network, and this effect becomes more important as the number of processors grows.

Recently, new coherence protocols have been proposed to address these problems [2, 3]. In particular, Virtual Hierarchies (VHs) [3] focuses on supporting a consolidated environment. Two different two-level coherence protocols (named VH_A and VH_B) that adapt to the VMs running in the server are presented.

The first level (intra-VM level) of these protocols handles most of the coherence actions, and the second level (inter-VM level) is only used for inter-VM sharing and dynamic reallocation of resources to the VMs. The authors claim that their best VH based protocol performs 12-58% better than the best alternative flat directory protocol.

VHs were designed for scenarios in which most sharing takes place within the VMs. Nevertheless, they are prepared to support memory deduplication [4], which is a very common technique in consolidated servers and the main source of interference among VMs. However, their performance when using memory deduplication has not yet been evaluated. This is one of our main purposes.

The simplest example of deduplicated memory is that of the code of the operating system. Using deduplication, if 16 VMs execute their own instance of Solaris 10, a single copy of the code would be present in memory thanks to the action of the hypervisor. Nevertheless, the code of other applications and data is often also deduplicated. Details on the deduplication process can be found on Section V.

We have reimplemented the proposed VH protocols as described in [3] and [5] and we have found that the implementation of the protocols becomes unexpectedly complex to be able to properly support data sharing among VMs. In particular, VH_B, as described in [3] and [5] is not deadlock-free. Moreover, when deduplication is used, the performance of these protocols is very close to that of a flat directory protocol, due to some extra management for shared blocks that must be carried out.

In this paper we analyze these protocols and their problems, and propose solutions for fixing them. In order to prevent deadlocks, we designed a solution based on time-outs. We also elaborate on the token counting mechanism used by VH_B and fix some of its drawbacks. We also demonstrate that, when deduplication is used, the benefits of these protocols in terms of performance are reduced.

The rest of the paper is organized as follows. Section II gives some background on VHs. Section III shows the deadlock problems found on the implementation of the VH_B protocol, and describes the solution that we have designed to prevent them. In Section IV, other features of VH_B

1550-6533/10 $26.00 © 2010 IEEE
DOI 10.1109/SBAC-PAD.2010.31

such as token management and private data optimization are discussed. Section V presents the new features that we have added to our simulator in order to model deduplication in a consolidated server. In Section VI we show the tests performed and their results. Finally, our conclusions can be found in Section VII.

II. Background

Next we discuss the base architecture used, the intra-VM level of the VH protocols, and the two different inter-VM levels of the protocols, that is, VH_A and VH_B.

Base Architecture: We focus on tiled-CMPs with a directory-based coherence protocol. This kind of chip is built by replicating basic constructing blocks containing a processor, an L1 cache, an L2 cache bank and a network interface. The L2 cache in the chip is shared among all tiles. Each memory block is mapped to its home L2 bank by using some bits in its address. When a block is present in any cache on the chip, its home L2 stores its directory information. When an L1 miss occurs, a request is sent to the home L2 bank. There is no need to keep directory information in memory.

L1 and L2 caches are non-inclusive. The directory information in L2 is stored in a different structure than the data blocks. Both structures have the same number of sets, but the associativity is bigger for the directory. When a block is evicted from its home L2 bank, the directory information remains. When a directory entry is evicted, the block is also evicted (if present), and every copy of the block is invalidated.

Virtual Hierarchy Intra-VM Protocol: The intra-VM protocol of the hierarchy, which is common for both VH_A and VH_B, is used to minimize the average L2 access latency. In order to do this, each VM is given a share of the L2 cache of the chip, preferably the nearest banks to the cores executing the VM. This reduces the distance that the misses need to travel due to the indirection introduced by the directory, which is located in the L2 banks belonging to the VM. This also isolates the coherence actions of each VM at this level of the protocol.

In order to find the home L2 for a block, every L1 cache has a dynamic table, mapped by some bits of the address of the block, storing the address-to-home-L2 correspondence. The table has as many entries as there are tiles in the chip (for the case when there is a single VM running in all the tiles or there is no virtualization at all). The L2 banks belonging to the VM are interspersed in the table. Reconfiguration of resources is possible by changing the contents of the table.

$\mathbf{VH_A}$ Inter-VM Protocol: This second level of the protocol keeps the coherence among VMs. When an L1 miss cannot be solved by the home L2, a request is sent from the home L2 to the directory in memory. Notice that, contrary to the base architecture explained before, directory information in memory is needed. A directory entry must be able to point to any subset of the L2 banks in the chip since, due to the possible reallocation of resources, any L2 can be the home L2 for any block.

In terms of performance, VH_A is equivalent to giving a private L2 cache to each VM. The advantage that it provides is the flexibility to reallocate the resources thanks to the dynamic home tile tables.

$\mathbf{VH_B}$ Inter-VM Protocol: VH_B is different from VH_A since it uses a broadcast-based second level protocol and mixes directory and token coherence. This allows the reduction of the directory information in memory to a single bit per block that indicates whether the memory holds the block and all the tokens for that block or not. When a request cannot be satisfied by the intra-VM level and comes to the inter-VM level, if the bit is active, then memory can directly answer the request by sending the data and all the tokens. Otherwise, a broadcast message that reaches every L1 and L2 cache is sent. Only the caches holding tokens have to answer that message in order to satisfy the request. Two versions of VH_B protocol have been proposed. They differ in whether broadcasts can bypass the L2 directory [3] or must be handled by it before reaching L1 caches [5]. The second alternative is not affected by all the deadlock scenarios, as we will see in Section III. On the other hand, the first alternative offers better performance since bypassing the L2 directory avoids one hop when solving inter-VM misses.

The L2-bypassing broadcast mechanism is supposed to simplify the protocol since many transient states in the L2 directory are no longer needed as every inter-VM request reaches every cache, and the token counting rules avoid any coherence violations. Additionally, the directory in L2 is inexact (because the broadcast messages bypass the L2 directory and therefore it no longer needs to hold precise information about the sharers). All of this allows several optimizations like local replacements for private data and directory victimization without invalidating the copies of the block.

Although the paper by Marty et al. [3] states for simplicity that the first level coherence protocol is the same in VH_A and VH_B, the use of broadcast and tokens in VH_B implies important changes in both levels of the protocol. First, since VH_B uses tokens, we need to keep the token count for the data in the cache. Second, the token counting rules force the L1s to hold at least one token to be able to access the data. In some cases, this prevents the home L2 from sending the data to the L1 requestor, at the intra-VM level, due to lack of enough tokens.

Further details on these protocols (VH_A and VH_B) can be found in [3] and [5].

A. VH_B and Token Coherence

Token-based and directory-based coherence mechanisms have already been successfully combined in a single protocol [6]. Therefore, it is reasonable to wonder whether VH_B, which uses token counting in addition to the directory information, is also a token-based protocol with a correctness substrate that would prevent any deadlock or starvation situations. However, its authors explicitly claim that unlike token coherence, VH_B does not require persistent requests to maintain liveness in the system [5]. Actually, they only use token counting to enable the reduction of directory information in memory to a single bit per block and allow tiles without tokens to avoid answering broadcast requests.

If the full correctness substrate of a token protocol were used, none of the problems that we have detected in VH_B's operation would be applicable. This way, VH_B would become nothing but a *performance policy* [7] for token coherence, and the resulting protocol would present higher hardware overhead due to the additional structures needed by the correctness substrate of token protocols. Additionally, the interactions between the token substrate and the directory information used by the protocol would need to be defined, like the actions that should be performed on the directory information when persistent requests need to be issued (for example, flushing that information).

In Section IV-A we discuss our approach to the token management in VH_B.

We find our dissertation on deadlock problems necessary as these are the problems that, in general, may arise in a hierarchical protocol with no single ordering point. Additionally, we present modifications to ensure the correctness of VH_B in Section III-B.

III. VH_B Problems Causing Deadlocks

In this section, we present a number of scenarios where the original design of VH_B would fail. This way, we show that the protocol, as described in [3] and [5], does not have the property of correctness and some modifications or additional mechanisms are needed.

We have analyzed VH_A and VH_B, and we have found that there are two root problems with the behavior of VH_B. The first problem is caused because invalidations sent (by broadcast) by the global directory do not wait for an acknowledgement. Hence, they can be delayed by the interconnection network and arrive after the original request has been satisfied. This means that they can arrive virtually at any time, hence they can interfere with later requests in unpredictable ways. The second problem, which is only present in the faster version of VH_B [3], is that there is no ordering enforced between inter-VM requests and intra-VM requests. This allows an intra-VM request to interfere with an inter-VM request (and conversely), making it fail.

A. Deadlock Scenarios

The first deadlock scenario is caused by the lack of acknowledgement to broadcast messages. When the directory broadcasts a request for a cache block, only the tiles that hold tokens for that block answer the request. When the request is solved, some of the broadcast request messages may have not reached its destination yet (e.g. they may have been delayed in the interconnection network). We call them *delayed request messages*. When a new write request is broadcast by the directory, some delayed messages from previous write requests may still be present in the network. Therefore, the broadcast messages from the current write request may not collect all the tokens since delayed request messages can "steal" some of the tokens. If the L1 cache that originated the delayed request messages has also issued a new write request, it will accept the tokens that its delayed request messages managed to collect. In this scenario, no L1 cache is able collect all the tokens and no write request can be completed.

Additionally, some other related minor issues were detected while implementing VH_B. For example, delayed request messages can also unintentionally solve a new cache miss. For this reason, a blocked L2 or the directory can receive an unblock message corresponding to a different request than the one they are serving. The protocol must deal carefully with this (the unblock message cannot be ruled out nor processed) or new deadlock scenarios can occur.

The second problem shows up when no ordering point between inter-VM and intra-VM requests is forced [3], therefore allowing the protocol to solve inter-VM requests with one less hop. Next we describe one example of an inter-VM request preventing an intra-VM request from succeeding, and then one example of an intra-VM request interfering with an inter-VM request.

In the example of Figure 1, L1a, L1b and L2 are assigned to VM0. *AnotherVM* represents a different VM that sends a request via the directory and gets the corresponding answer.

Let us suppose that one L1 holds all the tokens (L1b), and a different L1 in the same VM (L1a) issues a GetX (1). This request reaches the L2 and then is forwarded to L1b (2) because it is an intra-VM request. Before this forwarded GetX reaches L1b, an inter-VM request originated in a different VM is broadcast by the directory and reaches L1b (3). In response to the inter-VM request, L1b, which holds all the tokens, sends the data and a token to the other VM (4). When the intra-VM request reaches L1b (2), the response (5) contains all the tokens minus one. Hence, L1a will lack a token even after receiving the answer from L1b (5). There is no mechanism described in [3] to prevent such a simple case of deadlock.

In the same manner, an intra-VM read request can obtain a single token, preventing an inter-VM write request from collecting all of the tokens. Figure 2 depicts this situation. The broadcast messages sent by the directory (1, 2) cannot find the token that is sent from L2 to L1 (3, 4, 5) as a result of a read request. Therefore, the write request forwarded by the directory is never solved since it lacks one token.

Unfortunately, the absence of an ordering point among local and global requests creates so many different race

Cache States	
L2	
ILX	Block is not present, but local exclusive exists in the VM
IFX	Blocked, forwarded local request to local exclusive
L1	
NP	Block is not present (or was invalidated)
IS	Issued GetS
IM	Issued GetX
OM	Waiting for tokens in order to write the block
E	Block is in exclusive state
S	Block is in shared state
Message	**Description**
GetX	Write request
GetS	Read request
DataSh	Response message containing data and a token
DataOw	Response message containing data and every token but one

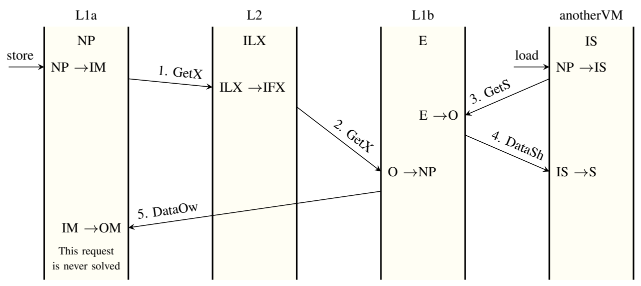

Figure 1. Deadlock. An inter-VM request makes an intra-VM request fail

Cache States	
L2	
S	The L2 holds the block and a token
SS	A load request is being served
ILS	Block is not present, but local sharer exists in the VM
L1	
NP	Block is not present (or was invalidated)
IS	Issued GetS
O	Owner
OM	Waiting for tokens in order to write the block
S	Block is in shared state
Message	**Description**
GetX	Write request
GetS	Read request
Unblock	Unblock to L2
DataSh	Response message containing data and a token

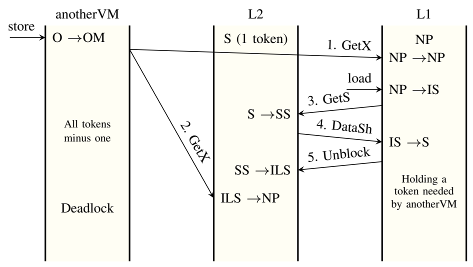

Figure 2. Deadlock. An intra-VM request makes an inter-VM request fail

conditions (we can find examples as complex as we want) that they cannot be considered in the protocol on a case-by-case basis.

Since both versions of VH_B [3, 5] present deadlock scenarios, we have decided to implement a simple and generic timeout mechanism, valid for either version of VH_B, to detect these potential deadlocks where a request cannot be solved. For evaluation purposes, we have chosen the faster version of VH_B [3], since the avoidance of a hop when solving inter-VM requests is very beneficial when memory deduplication is on.

B. Our Solution to Deadlocks

In order to prevent deadlocks, we have designed a timeout mechanism that consists of timers placed in each L2 cache bank and in the directory in memory and that can trigger persistent requests in some very infrequent situations.

Each time the directory broadcasts a request it blocks and sets a timer. If the request timeouts before the directory is unblocked, then the directory rebroadcasts the request and resets the timer.

The timeouts in L2 are meant to turn intra-VM requests into inter-VM requests by sending them to the directory. Each time the L2 receives a request and tries to solve it inside the VM, a timer is set for the request. When the request timeouts, it is sent to the directory, becoming an inter-VM request. Hence, a request in the L2 can only timeout once. After that, the directory will always solve the request. We empirically determined a 3000-cycle threshold for the timeouts. The main characteristics of the timeout mechanism are summarized in Table I.

With this, we force an ordering point in the directory for requests that cannot be successfully completed by the protocol's regular operation. However, the broadcasts sent by the directory can always fail unpredictably due to successful intra-VM requests. For example, there is a risk of starvation when a block is in exclusive state in a VM. This block can travel from one L1 cache to another inside the VM over and over again, due to intra-VM requests. The broadcasts of an inter-VM request would be useless if they always reach the caches in an inadequate order (an order that we cannot control). Although this is extremely unlikely, the L1 that issued the inter-VM request would suffer from starvation.

To account for such an infrequent situation in which a inter-VM request is never satisfied, we can afford to use a drastic mechanism which consists of using a persistent request, like those used in the banked arbitration scheme of token-based protocols [8], after a number of retries for the request have been performed. We fix one memory controller as the only arbiter in the chip, and, different to current token coherence implementations, we allow only a single persistent request active at a time in the chip. This scheme scales well because only a single register in each cache is needed to store the address of the block that caused the only active persistent request, regardless of the number of processors in the chip. The situation that activates a persistent request is so infrequent that it has never taken place in our simulations of consolidated workloads. Therefore, contention is not a real

Table I. TIMEOUT MECHANISM

Timer Location	Event that activates a timer	Event that deactivates a timer	Timeout effect	The timer is activated again after a timeout
L2 Cache	The L2 tries to solve an intra-VM request.	An unblock message for the request reaches the L2.	The request is forwarded to the directory (it becomes an inter-VM request).	No.
Directory	A broadcast is sent to solve an inter-VM request.	An unblock message for the request reaches the directory.	The request is rebroadcast.	Yes.
Directory	A broadcast is sent to find a cache to send writebacked tokens to.	A message from a cache that can accept tokens arrives.	The find message is rebroadcast.	Yes.

problem despite having a single arbiter and allowing a single persistent request.

IV. CLARIFYING OTHER FEATURES OF VH_B

A. Token counting

When a protocol uses tokens, whose number is limited (so that all elements can have at least one token simultaneously), a mechanism is needed to ensure that every read request eventually gets at least one token and that every write request eventually gets all of them. In the latter case, VH_B *tries* to ensure it by blocking the directory and sending a broadcast message that reaches every possible holder of tokens, although, as seen in Figure 2, it does not actually ensure it. However, for a read request, there is no mechanism described to ensure that the requestor will eventually get a token. Moreover, there is no definition about who sends the token needed by a read request. The cache holding the owner token is a good candidate for this task, and we use it in our implementation.

In order to keep the protocol simple, the owner cache should be careful not to run out of regular tokens, or we would be forced to implement some mechanisms to collect tokens which would increase the complexity of the protocol (like persistent requests do in other token based protocols).

The simplest way to avoid this problem consists of making the cache holding the owner token send a single token to each read request. Contrary to VH_A, where once an L1 is a sharer every read request can be satisfied inside the VM, load requests in VH_B need a token, and if no writebacked tokens exist in the home L2 for that VM, the request must reach the cache holding the owner token. This increases the average latency of this kind of requests.

Additionally, the original authors [5] propose a token coalescing mechanism to deal with L2 writebacks since the directory has only one bit to encode token information. When the directory receives a writeback from L2, it broadcasts a FIND message that is answered by the caches that hold tokens. After receiving the answers, the directory sends the tokens to one of those caches. This can lead the L1 cache holding the owner token to run out of regular tokens, since other caches in the system can hold more than one token each.

Moreover, when the directory sends the owner token to an L1 as a consequence of a writeback from an L2, the home L2 for that L1 is not informed unless mechanisms such as hints are used. Hence, an L2 must expect new kinds of block replacements from L1 apart from the ones that can be expected based on the (now incomplete) directory information.

In order to keep the protocol simple, we use a different approach. In our version, when some tokens are writebacked to the directory, the FIND message that the directory sends is answered only by the L1 holding the owner token. If the owner token is also writebacked to the directory, then the directory issues a different type of FIND message which is answered by any cache that holds tokens. In the latter case, the cache that receives the tokens becomes the owner.

In addition, under some circumstances (like a concurrent writeback that contains the owner token to L2 cache), no cache might answer the FIND message. Hence, we extend the original mechanism with timeouts to rebroadcast the FIND messages. After a number of unsuccessful retries, the persistent request described in Section III-B is issued to force all tokens to be sent to the directory.

With respect to the simplicity, due to all the problems with VH_B explained so far (inexact directory, deadlocks, broadcasts that bypass the L2, token counting problems), every cache can receive almost any type of message at any time while being in any state, making the number of possible transitions of the protocol increase dramatically. This makes reasoning about the protocol difficult, and it is contrary to the statement of its authors that VH_B is a simple protocol since many transient states disappear. This also makes possible for a cache to receive an unexpected message containing tokens. When such a thing happens, we forward the tokens to the directory.

B. Private Data Optimization

The private data optimization used in VH_B, which sometimes provides a great performance improvement [3], has several problems. There are two main concerns: unsuccessful private data retrieval and finding private data. We explain them next.

Private data retrieval: The private data optimization tries to improve performance by bringing the private data to the L2 of the tile that the L1 which is using that data belongs to. This implies that the home L2 does not know the location of such data. When a core wants to access one block and the L1 holds neither the data nor a token, then, instead of directly accessing the home L2 for the block, the L1 tries to recover the private data from its tile's L2. If it succeeds, the miss is solved without needing to go out of the L1's tile. If not, the request is forwarded to the home L2, resulting in an additional hop (the one performed within the tile to try to recover the private data). Notice that the access to the L1's tile's L2 could be performed in parallel to the access to the home L2, avoiding the extra hop. However, if the data were found in the L1's tile's L2 and the home L2 were accessed in parallel, the home L2 and the memory directory would block, since they would try to locate the data, and an unnecessary message would be broadcast. This would add extra latency to other requests that would have to wait until the L2 and directory unblock, and would also increase the traffic due to the extra broadcasts.

When the private data retrieval succeeds, in general, no hops are avoided since the data would also be found in the home L2 if the optimization were not used. Nevertheless, this optimization reduces the latency of these misses because the L2 where the data is found is closer than the home L2.

Unfortunately, many blocks are not private to the tile and their information is in the home L2. In that case, this optimization introduces an extra hop to check if private data is available in the tile's L2. In particular, this would happen with deduplicated data.

The hit/miss ratio of private data retrievals affects the performance. In Section VI we show that this ratio is very low for many applications.

Finding Private Data: Another problem of the private data optimization is the need to find private data held by an L1 cache by means of broadcast, even when the access is performed by an L1 in that same VM. This problem arises because, when an L1 cache holds private data, the home L2 has no directory information about the data.

Without the optimization, an L1 miss would find the information about the owner in the home L2, and would be solved within the VM. With this optimization, this information is not available in the home L2 and the miss has to rely on a broadcast performed by the directory.

When both private data retrieval and finding private data problems show at the same time, a two-hop miss in the original VH_B becomes a five-hop miss plus broadcast in the optimized version of VH_B. This increases both miss latency and network traffic.

When private data is found by a read request and becomes shared data the unblock message which is sent to the home L2 contains the identity of the owner. This allows subsequent read requests in the VM to hit in the home L2.

V. Simulation Infrastructure

We use our Virtual-GEMS [9] simulator to perform our evaluation. Virtual-GEMS is based upon GEMS [10] and Simics [11], and simulates a consolidated server using virtualization. To perform this work, we have added some new features to the simulator. The main one is the ability to simulate memory deduplication.

With deduplication, when one VM first accesses a memory page, its *real address* is mapped to a *physical address* in memory. In this moment, the hypervisor checks whether the new memory page is shared with some other VMs. In order to do this check, the MD5 hash for the new page is calculated. The contents of the page are then compared to the ones of the pages with which it shares its hash. If there is a match, the identical pages will share the same physical address (and only one of the pages will be used for future content comparisons). Otherwise, the hash is stored and the page is mapped to a new location in physical memory. A copy-on-write policy is used when shared pages are written. This scheme is similar to content-based page sharing as implemented in VMware [4], Disco [12] or the Linux kernel.

Broadcast support is needed in order to prevent VH_B from congesting the network when memory deduplication is used. If no broadcast support exists, the network interface of the memory controller splits each broadcast in at least 64 unicast messages to reach every L1 and L2 cache in a 64-core CMP. All of these messages must traverse the same first link, one at a time. Even though there are several memory controllers in the chip (each with its own network interface) and broadcasts are distributed among them, VH_B without broadcast support is orders of magnitude slower than the rest of the protocols tested, and there is no use in further analyzing its results. This also makes impractical to use acknowledgements for broadcast requests, since this would require at least 64 unicast messages, coming from different sources, trying to traverse the same link to reach their destination.

In order to use a detailed network and model the effect of multicast and broadcast messages, we have implemented multicast support in the Garnet [13] network simulator included in GEMS. At the cost of extra hardware, we provide the router microarchitecture with the ability to route a multicast packet to several output ports. The input buffer containing the message is set free after the message has traversed every output link towards its destinations.

VI. Evaluation and Results

The characteristics of the simulated system are shown in Table II.1. We use as a base case a flat directory like the one discussed in Section II. The benchmarks used can be seen in Table II.2. Three different VH based protocols are tested: VH_A, VH_B and VH_B-*opt* (that is, VH_B with the private data optimization). We use a MOESI state scheme in all the protocols.

Figure 3 and Table III summarize the results of our experiments. In Figure 3.1, we can see that when not using deduplication, the best performing protocol is VH_B which performs 22.6% better than the flat directory. However, the performance of this protocol with the private data optimization (VH_B-*opt*) decreases. This is caused by the extra broadcasts to find the private data belonging to another core (whose effects are captured by the Garnet network), as well as by the extra hops caused by the low hit rate of the cores accessing to their tile's L2 to get private data (Tables III.4, III.5 and III.6).

Table II. SYSTEM AND BENCHMARK CONFIGURATIONS.

II.1. System configuration	
Processors	64 UltraSPARC-III+ 3 GHz. 2-ways, in-order.
L1 Cache	Split I&D. Size: 64KB. Associativity: 4-ways. 64 bytes/block. Access latency: 1 (tag) + 2 (data) cycles.
L2 Cache	Size: 256KB each *slice*. 16MB total. Associativity: 8-ways. 64 bytes/block. Access latency: 4 (tag) + 6 (data) cycles.
RAM	4 GB DRAM. 8 memory controllers along the borders of the chip. Memory latency 300 cycles + on-chip delay
Page Size	4 KB
Interconnection	Bidimensional mesh 8x8. 16 byte links. Latency: 4 cycles per link.

II.2. Benchmarks			
Workload	**Description**	**Size**	**Simulation**
apache4x16p	Web server with static contents	3000 transactions per VM	Four 16-processor Apache VMs
jbb4x16p	Java server	5000 transactions per VM	Four 16-processor JBB VMs
barnes4x16p	Simulation of gravitational forces	8192 particles	Four 16-processor Barnes VMs
radix4x16p	Sorting of integers	1M integers	Four 16-processor Radix VMs
unstructured4x16p	Computational fluid dynamics application	Mesh.2K, 5 time steps	Four 16-processor Unstructured VMs
LU4x16p	Factorization of a dense matrix	512x512 matrix	Four 16-processor LU VMs
water-nsq4x16p	Molecular dynamic simulation of water	512 molecules	Four 16-processor water-nsq VMs

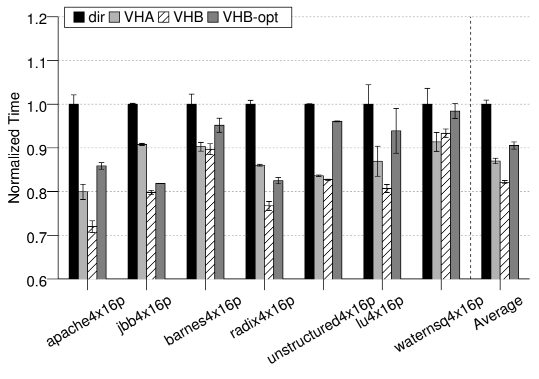

Fig 3.1. Deduplication is off

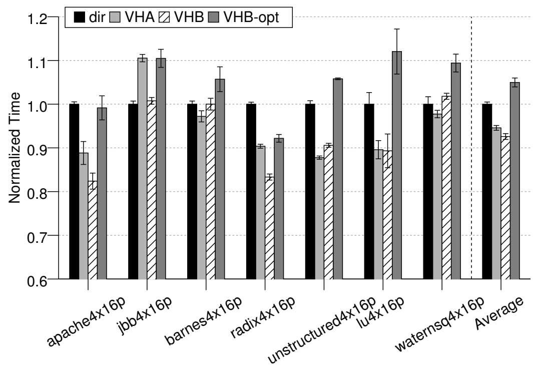

Fig 3.2. Deduplication is on

Figure 3. Performance of the tested configurations.

Table III. STATISTICS OF THE SIMULATIONS.

	Apache	**JBB**	**Barnes**	**Radix**	**Unstr.**	**LU**	**Water**	**average**
III.1. Performance increase when deduplication is on								
Dir	10.1%	19%	6.6%	3.7%	2.9%	−0.8%	4.6%	6.6%
VH_A	−0.9%	−2.3%	−1%	−1.3%	−2.1%	−3.9%	−2.2%	−2%
VH_B	−3.9%	−6.1%	−4.6%	−4.7%	−6.4%	−11.6%	−4.3%	−5.9%
VH_B-*opt*	−4.9%	−13.3%	−4.2%	−7.8%	−7.1%	−20.3%	−6.2%	−9.1%
III.2. Network traffic increase when deduplication is on								
Dir	1.6%	−8.4%	0.4%	2.8%	−1.7%	−1.1%	−2.8%	−1.4%
VH_A	0.2%	−10.5%	2.5%	−1.1%	−3%	2.6%	0.7%	−1.3%
VH_B	10.7%	68%	11.1%	9.4%	2%	83%	17.8%	28.9%
VH_B-*opt*	10.3%	79.7%	14.1%	11.1%	−9%	48.8%	14.9%	25.4%
III.3. Average memory saving thanks to deduplication								
	21.8%	50.9%	13.8%	15%	20.5%	33%	28.4%	26.2%
III.4. L1 misses that required a broadcast (no deduplication)								
VH_B	1.9%	1.1%	5.6%	2.32%	0.9%	0.3%	3.2%	2.2%
VH_B-*opt*	6.1%	1.9%	6.5%	13.8%	1.2%	6.7%	6.7%	6.3%
III.5. L1 misses that required a broadcast (deduplication)								
VH_B	4.1%	26.8%	6.9%	4.4%	1.4%	20.5%	6.7%	10.1%
VH_B-*opt*	8.3%	26.8%	7.8%	16.7%	1.5%	24.5%	10.3%	13.7%
III.6. VH_B-*opt* L1 misses that found private data								
No Dedupl	8.3%	16.2%	3.4%	18.1%	1.2%	4.9%	7.1%	8.4%
Dedupl	7.3%	7.4%	1.6%	16.7%	0.9%	1.3%	3.7%	5.6%

On the other hand, when deduplication is used, the performance advantage of VH based protocols gets reduced (Figure 3.1). VH_B is still the protocol with the best performance, but now it is only 8.7% faster on average than the flat directory. VH_B-*opt* performs even worse than the flat directory in most benchmarks due to the problems noted in Section IV-B. Table III.6 shows the low hit rate of private data retrievals. This is the most likely scenario in a consolidated server.

The main reason for this is that Dir performance improves 6.6% thanks to the L2 savings provided by the deduplication of data (the L2 miss rate of Dir decreases an average 9% when deduplication is used). Since VH protocols duplicate the deduplicated data again in L2, their performances do not improve (their L2 miss rate remains the same); and since the use of the inter-VM level of the protocol is needed to access shared data, the performance of VH_A and VH_B actually decreases. This is especially noticeable in the VH_B case, whose performance is significantly affected by the broadcasts needed to locate shared data (see the contrast between Tables III.4 and III.5).

Despite the support for multicast added to the network, which is needed only by VH_B and VH_B-*opt*, the network usage of this protocol is noticeably higher when deduplication is used, due to the frequent broadcasts (Table III.2).

Our tests also show how a very small percentage of the intra-VM requests of VH_B timeout and must be forwarded to the directory in order for them to complete. An even smaller percentage of the requests needs to be rebroadcast. This only happens when deduplication is used. It does not noticeably affect performance, but shows how the mechanisms that we have added are actually needed for correctness.

VII. Conclusions

Server consolidation is an increasingly important technique to make the most out of the new architectures that provide a high number of processors. The idea of adapting cache coherence protocols to the needs of server consolidation is very compelling.

The first proposal in this direction was VHs [3]. Our tests confirm the statement from the original authors that their protocols perform considerably better than a flat directory coherence protocol, but only as long as each VM is totally isolated and the inter-VM protocol is barely used.

However, a commonly used technique in consolidated environments is memory deduplication. We show that, when memory deduplication is used, the performance of a flat directory protocol improves 6.6% due to the L2 cache space savings provided by the deduplicated data. On the contrary, VH based protocols cannot take advantage of this circumstance since deduplicated data is replicated again in L2 cache. This way, the performance of the flat directory approaches that of VHs. Moreover, the most innovative proposed protocol, VH_B, needs broadcast support in order to avoid congesting the network when using deduplication.

Furthermore, VH_B needs additional mechanisms not described in the original proposal to ensure correctness, since the mixing of directory-based and token-based coherence, and the absence of acknowledgements and serialization points for the broadcast requests make the protocol prone to deadlock. Our solution for this is to use a carefully designed mechanism to manage the tokens and to use a timeout mechanism to reissue requests. For the rare scenarios in which a request can suffer from starvation we propose a single persistent request mechanism.

Additionally, we show that the private data optimization proposed for VH_B is not always beneficial, and it causes performance degradation for some applications due to the small hit rate shown and the extra broadcasts needed to locate private data.

All of this reduces the attractiveness of VHs. Nevertheless, the path started by VHs looks very promising, and the drawbacks of these protocols, pointed out in this paper, should be addressed by new proposals that adapt better to a consolidated server environment.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. This work has been jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 00001/CS/2007, and also by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2009-14475-C04-02". Antonio García-Guirado is also supported by a research grant from the Spanish MEC under the FPU National Plan (AP2008-04387).

References

[1] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood, "Improving multiple-cmp systems using token coherence," in *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA)*, 2005, pp. 328–339.

[2] A. Ros, M. E. Acacio, and J. M. García, "DiCo-CMP: Efficient Cache Coherency in Tiled CMP Architectures," in *22nd Int. Parallel & Distributed Processing Symposium (IPDPS)*, 2008, pp. 1–11.

[3] M. R. Marty and M. D. Hill, "Virtual Hierarchies to Support Server Consolidation," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 46–56.

[4] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," in *5th Symposium on Operating System Design and Implementation (OSDI)*, 2002, pp. 181–194.

[5] M. R. Marty, "Cache coherence techniques for multicore processors," PhD in Computer Science, University of Wisconsin - Madison, 2008.

[6] A. Raghavan, C. Blundell, and M. M. K. Martin, "Token tenure: Patching token counting using directory-based cache coherence," in *Proceedings of the 41st Annual International Symposium on Microarchitecture (MICRO)*, 2008, pp. 47–58.

[7] M. M. K. Martin, "Token coherence," PhD in Computer Science, University of Wisconsin - Madison, 2003.

[8] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: decoupling performance and correctness," in *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*, 2003, pp. 182–193.

[9] A. García-Guirado, R. Fernández-Pascual, and J. M. García, "Virtual-GEMS: An Infrastructure To Simulate Virtual Machines," in *Proc. of the 5th Int. Workshop on Modeling, Benchmarking and Simulation (in conjunction with ISCA)*, 2009, pp. 53–62.

[10] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, November 2005.

[11] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[12] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum, "Disco: Running commodity operating systems on scalable multiprocessors," in *ACM Transactions on Computer Systems (TCS)*, 1997, pp. 143–156.

[13] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 33–42.

[14] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.