Scalability of Avoidance-Based Transactional Cache Coherency

Gareth Hughes

Department of Computer Science
Australian National University
Canberra ACT 0200 Australia

Email: huggchon@mehta.anu.edu.au

1 Introduction

The scalability of cache coherency schemes plays a major role in the overall performance of any scalable distributed system that utilises client data caching. In particular, transactional cache coherency algorithms must be scalable if the systems that are built around them, such as persistent object stores, are to be scalable.

The avoidance-based transactional cache coherency [Hughes 1998] algorithm PS-AA [Carey et al. 1994] was used as the basis of an investigation into such scalability. This particular algorithm was chosen as it is considered the state of the art in avoidance-based algorithms.

This paper discusses the PS-AA algorithm, presents the scalability experiments and examines the results that were obtained.

1.1 The PS-AA Algorithm

The PS-AA algorithm behaves like a standard page server algorithm in the absence of conflicts [Carey et al. 1994]. However, as the algorithm can perform both callbacks [Hughes 1998] and locking in an adaptive manner, when data conflicts arise the algorithm can de-escalate to a finer-grained operation as needed. Furthermore, the algorithm can “re-escalate” to the original grain when the conflict that caused the de-escalation to occur has been resolved [Carey et al. 1994].

The following sections give a closer look at how the algorithm performs read and write operations. This allows an exploration of some of the finer points of the PS-AA algorithm, as some parts are quite complex.

1.1.1 Read Requests

When a client wants to read an object that is not present in its cache, or is marked as unavailable, it must send a read request to the server owning the object. When a server receives such a request, it checks for conflicts on both the object being requested and the page it lies on. There are three cases that can arise from this check [Carey et al. 1994; Hughes 1998]:

No Conflict  No client has a page-level write lock on the page, and no client has an object-level write lock on the object itself, and so the client is free to read the object. In this case the PS-AA algorithm sends the entire page back to the client with any objects currently write locked by other clients marked as unavailable.

Object-Level Conflict  When another client has already been granted an object-level write lock on the requested object, the read request must block until this write lock is released.

Page-Level Conflict  If the read request conflicts with a page-level write lock, the client holding the write lock is asked to de-escalate its lock to a finer (object) grain. The client does this by obtaining object-level write locks for all objects on the page it has updated since being granted the page-level write lock. After this de-escalation is done, the server then checks for conflicts again, proceeding according to one of the two previous cases.

1.1.2 Write Requests

When a client wants to update an object for which it does not have write permission, at either an object or page level, it must send a write lock request to the server owning the object. There are again three cases that can arise when the server checks for conflicts on the object and page it resides on [Hughes 1998]:
No Conflict  No client has a page-level write lock on the page, and no client has an object-level write lock on the object itself, and so the client is free to update the object. Callbacks are sent to all other clients caching the page. When clients receive such a callback, they attempt to purge the entire page from the cache. When the server receives all of the callback acknowledgements, there are two possible outcomes: if the page has been invalidated everywhere, a page-level write lock can be granted to the client. Otherwise, objects on the page are in use and thus only an object-level write lock can be granted to the client.

Object-Level Conflict  When another client has already been granted a write lock on the requested object, the write request must block until the conflicting lock is released.

Page-Level Conflict  If the write request conflicts with a page-level write lock, the client holding the write lock is asked to de-escalate its lock to a finer (object) grain. This is done in exactly the same way for read requests — that is, the de-escalating client obtains object-level write locks for all objects on the page it has updated since being granted the page-level write lock. After this de-escalation is done, the server then checks for conflicts again, proceeding according to one of the two previous cases.

1.2 Distributed Deadlock Detection

As the experiments presented in this paper dealt with such a high level of scalability, a serious problem with traditional distributed deadlock detection algorithms was discovered. Standard timeout-based and edge-chasing solutions [Knapp 1987] could not handle systems with up to 128 nodes. In particular, the edge-chasing algorithms tended to flood the system with deadlock probes, essentially bringing the system to a halt under the sheer number of messages being sent.

To combat this, a new distributed deadlock detection algorithm was designed and implemented for the sake of the scalability experiments. This new algorithm, called lazy edge-chasing [Hughes 1998], compensated for the high degree of contention in the system by delaying the sending of probes until a short period of time has passed. This delay allows non-deadlock waits to continue on their own after the conflict is resolved, and resulted in a great increase in overall performance of the experimental system.

2 Experimental Framework

The platform for the experiments is the PS-AA cache coherency algorithm implemented as part of a scalable high performance transactional storage layer currently under construction. The architecture of the storage layer is typical of adaptive page shipping client-server object cache architectures. The distinguishing feature of the system is that it is “serverless”. Each node undertakes client and server roles, operating in a client-peer environment [Carey et al. 1994].

2.1 Workloads

The HOTCOLD workload [Carey et al. 1994] was adopted as the basis for the scalability experiments experiments. This workload was chosen both because of its suitability for modelling low to moderate levels of contention in a page serving object cache architecture and because of its familiarity. The HOTCOLD workload models contention by assigning each client a disjoint range of pages within the database which that client will tend to access most frequently, called a “hot range”. The remainder of the database is referred to as that client’s “cold range”. By varying the probability of hot and cold accesses, various levels of contention can be modelled.

The original workload outlined in [Carey et al. 1994] was implemented as closely as possible, in particular the hot ranges contain about the same number of objects as in the original (1024 versus 1000), so the level of contention will be similar for similar parameter settings. There have been, however, a number of minor adaptations. The most significant of these has been to place a stronger emphasis on the object as the unit of contention. Where transactions in the original workload consist of a series of page accesses each of which involve access to a number of that page’s objects, the transactions in these experiments consist of a series of object accesses, independent of which pages they are on.

Localised HOTCOLD workload  The first experiment, localised HOTCOLD, can be thought of as a best-case scenario for scalability. The hot range for each client is chosen so that it is “owned” by just one server (it was ensured that it was not the local server). In this situation, the effects of page caching come into play — that is, the more pages that are accessed from a single server the more likely it is that a requested object already lies in a client’s cache.
Scattered HOTCOLD workload  The second experiment, scattered HOTCOLD, can be thought of as worst-case. Here the hot range for each client is chosen so that the pages are distributed across all servers. The number of servers involved in each transaction is therefore generally high and is largely independent of the level of contention. This scenario thus tests the robustness of the cache coherency algorithm.

In both cases, the system throughput was measured against transaction size for different levels of contention. As the transaction size increases a higher probability of conflict and a larger number of servers involved in each transaction are seen.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
<th>Localised</th>
<th>Scattered</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatabaseSize</td>
<td>Size of database in pages.</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td>ObjectsPerPage</td>
<td>Number of objects per page.</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>HotBounds</td>
<td>Page bounds of hot range.</td>
<td>$p$ to $p+127$</td>
<td>$128p + n$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$p = 128n$</td>
<td>$p = (0, 1, \ldots, 127)$</td>
</tr>
<tr>
<td>HotAccessProb</td>
<td>Probability of accessing an object in the hot range.</td>
<td>0.6 to 1</td>
<td>0.6 to 1</td>
</tr>
<tr>
<td>WriteProb</td>
<td>Probability of updating an accessed object.</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>ThinkTime</td>
<td>Mean think time between transactions</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Workload parameters for 128 clients.

2.2 Hardware Platform

The platform for the experiments conducted was a 128-node Fujitsu AP1000 distributed memory multicomputer based on a worm-hole routed 2-D torus network. The user sees one-way latencies through MPI of $125\mu$sec (3125 25MHz SPARC-1 IU cycles) and bandwidth of 2.69MB/sec [Sitsky et al. 1994], giving an $n_{1/2}$ value $^1$ of 336 bytes. Each of the nodes on the machine has a 25 MHz SPARC-1 processor with 16 MB of RAM. The machine runs CELL-OS, a simple UNIX-like, single user, short-lived operating system.

3 Results

The PSI/PS-AA system was tested with both the localised and scattered HOTCOLD workloads. A wide range of values across the various dimensions of the HOTCOLD parameter space were used to give comprehensive scalability results. The results of these tests are summarised in figure 1 and figure 2.

The scalability results shown are in terms of the number of effective processors when all 128 processors are in use, for a range of test conditions. The effective number of processors is a measure of the operation throughput in the 128 processor case relative to the operation throughput in the base, or 2 processor, case. Perfect scalability would expressed as 128 effective processors. The graphs in figure 1 and figure 2 thus only compare the throughput of the 128 processor case with the base case.

![Figure 1](image1.png)

Figure 1: Scalability with localised hot ranges, contention increasing left to right, top to bottom.

$^1$The latency/bandwidth break-even point in message passing systems is referred to as the $n_{1/2}$ value.
The scalability graphs clearly show an inverse correlation between cold access probability and scalability. This is the result of bias in the workloads that favours cases where \( N \) is small; this occurs in both the localised and scattered benchmarks.

The lack of significant impact of hot range/server locality on scalability is an indication of the robustness of the PS-AA algorithm. The level of communication and number of servers in each transaction is generally high, and largely independent of the level of contention, in the scattered benchmark [Blackburn 1998]. The fact that the system handles this extra communication without an impact on scalability is perhaps the most important fact learned from these experiments [Hughes 1998].

### 3.1 Summary of Results

The graphs indicate that the PS-AA algorithm scales very well across a wide range of circumstances, irrespective of hot range/server locality, and is only sensitive to probability of cold accesses and transaction length. However, in-depth analysis of the workloads reveals that there are inherent biases that result in the scalability being understated [Blackburn 1998; Hughes 1998]. The lack of significant impact of hot range/server locality on scalability also indicates how robust the PS-AA algorithm is.

### 4 Conclusions

The experiments presented here explore the notion of scalability with regards to transactional cache coherency algorithms. This paper has given a demonstration of a fairly high degree of scalability by the PS-AA algorithm under a wide range of workloads conditions. This indicates the robustness of the PS-AA algorithm in the face of data contention, and the suitability of the system for a client-peer architecture.

### Bibliography


