Journal of Universal Computer Science, vol. 1, no. 6 (1995), 399-409
submitted: 31/3/95, accepted: 16/6/95, appeared: 28/6/95©Springer Pub. Co.

Distributed Caching in Networked File Systems

Artur Klauser
(Institute for Applied Information Processing and Communications,
Graz University of Technology, Austria,
aklauser@iaik.tu-graz.ac.at)

Reinhard Posch
(Institute for Applied Information Processing and Communications,
Graz University of Technology, Austria,
rposch@iaik.tu-graz.ac.at)

Abstract: Changing relative performance of processors, networks, and disks makes it necessary to reconsider algorithms using these three resources. As networks get faster and less congested topologies emerge, it becomes important to use network resources more aggressively to obtain good performance. Substitution of local disk accesses by accesses to remote memory can lead to better balanced resource usage and thus to faster systems. In this work we address the issue of file caching in a networked file system configuration. Distributed block-level in-memory caches are considered. We show that carefully constructed distributed concepts can lead to lower server load and better overall system performance than centralized concepts. Oversimplification, although aimed at gaining performance for single components, may deteriorate overall performance as a result of unbalanced resource usage.

Key Words: networked file systems, distributed file caches, load balancing, file system performance

Category: D.4.2, D.4.3, C.4

1 Introduction

File caching is used as the most important method to overcome the inherent speed difference between processor and disk. In centralized systems file caches form an intermediate storage level between slow disk storage and fast memory. File accesses in distributed systems are based on a client-server computing model [Coulouris, Dollimore 1988] and have to propagate through various instances. These present many opportunities for caching at various levels like the server, the client, or *the network*.

We compare a number of client caching approaches in distributed client-server systems. The performance of a single server cache with cacheless clients is used as a reference point for our comparisons. Our major goal is to find effective approaches for distributed caching which perform at least as well as centralized server caches, while at the same time reducing the load usually put on the server. For this considerations we assume standard Unix file system semantics in the distributed system. Client file caches are maintained in client memory, not on client disks. Caching and cache coherency is based on the file system block level. Clients reading a file see changes to this file immediately after completion of the write operation, not only after close.

To compare the performance of various algorithms we use trace-driven simulations. Traces come from a measured, real workload to prevent biasing the simulation with synthetic workloads that might make inadequate assumptions about the file access and file sharing profile. Simulations also let us lift restrictions typically found in current hardware and allow a prediction of the performance of future systems. Moreover, simulations allow exact repeatability, which is an important advantage for studying algorithms in distributed systems.

Our computing model is a general MIMD type architecture with distributed memory, and a point-to-point interconnection network. This model fits both, networks of workstations (NOWs) with high-performance communication networks (e.g. ATM, HIPPI) and MPP machines. The only means of inter-node communication is by message passing.

An important point of the network structure is its ability to perform multiple network operations concurrently. Distinct pairs of machines can communicate concurrently by means of different communication links. We assume point-to-point networks to be more appropriate for parallel and distributed systems, as they are more scalable than multidrop networks, which are limited to one network operation at a time.

As network speeds evolve rapidly we chose to simulate the internal communication network of an MPP supercomputer, namely an Intel Paragon. Although NOWs do not usually reach this performance yet, they will be able to operate at such speeds in the near future. The model uses a direct network with 2D-mesh topology and wormhole routing with an E-cube routing decision strategy (x-y routing). All message routing is implemented by dedicated hardware. The simulation model comprises of a 4×4 mesh with 16 processing nodes. Basic simulation parameters are 105 μs latency and 26 MBytes/s bandwidth, performance figures measured in Transmittal-8, an early beta release of the Paragon operating system.

[Section 2] gives an overview of the implemented caching policies. The simulator and the file system trace data are introduced in [Section 3]. [Section 4] discusses results of the simulations, [Section 5] presents related work, and the work is concluded in [Section 6].

2 Caching Policies

All caching models have a number of common features. Unless otherwise noted caches are assumed to operate at the file system block level, with a block size of 4-kByte and a write-back policy with server driven invalidations. A cache replacement strategy of least recently used (LRU) is used in all cases. Although this strategy is not optimal, many studies have shown that it is close to optimal ([Maffeis 1992],[Maffeis, Cap 1992]). Moreover, LRU guarantees the inclusion property, i.e. caches of size $i+1$ hold all items of size i caches, plus one additional item. This property is important for variable-size caches, as it guarantees monotony of the cache's hit rate function. Finally, all models use fully associative caches which give good performance and only require minimal additional overhead in the case of file accesses, which are rather expensive operations already.

2.1 Fixed-Size Caches

The following reference models are used in the evaluation process. These models have been chosen to give some upper and lower bounds on specific distribution concepts. They are not intended to be sophisticated implementations of these concepts. The order of presentation is approximately an order of increased complexity. A more detailed description of the models can be found in [Klauser 1994].

Server Cache Only (SCO): Server caching with cacheless clients is used as a reference point for comparing other policies. This model requires every client access to be forwarded to the server, resulting in substantial network and server load. It provides an upper bound on network traffic induced by the file system. The centralized design imposes strong limitations on scalability.

Local Disk (LOD): This model assumes each client to use a local disk for file storage and also assumes a memory cache to be operated by each client. The server's role is only that of a coordination instance, controlling the traffic flow between clients. Our interest is in the cache hit rates. As the client-server traffic does not contribute to this measure it has been omited. This model presents the most optimistic view of a completely distributed file service, where all accesses can be fulfilled locally. Any realistic implementation would also induce some client-server and client-client traffic.

No Coherency (NOC): With this model we assume a configuration with server and client caches. Compared to SCO the traffic on the network is reduced by the introduction of the additional caching level at the clients. Multi-client cache consistency is not modeled in this approach, thus reducing the network load to an absolute minimum. Only misses in the client cache and cache write-back operations generate traffic on the network. This approach presents an unrealistically optimistic network load. Realistic implementations would induce higher network load due to coherency traffic.

Write Through All (WTA): An implementation of the NOC approach with added coherency traffic is presented with this model. WTA uses the easiest way to guarantee consistency in the system, which is a write-through caching scheme. All changed blocks are transferred form client to server as part of the write operation. This guarantees that the server is always in possession of the most recent version of every block and thus can service requests from other clients with up-to-date data. As many files are only used by one client this protocol generates lots of unnecessary operations on the network and on the server. It is intended as a pessimistic model for guaranteeing global consistency on the block level. The amount of write traffic from clients to the server is the same as in the SCO model.

Write Share Sequential (WSS): Whereas WTA writes back blocks which could be kept locally without degrading client cache coherency, WSS seeks to eliminate this additional traffic. Analyzing file access traffic reveals that most of the written files are not actually shared between clients. Only a small fraction of files are actively shared. By using different write policies for shared and non-shared files the excess coherency traffic can be eliminated. WSS uses a write-back policy for non-shared files, which is dynamically changed to write-through as soon as file sharing is detected by the server. This guarantees a consistent view on the server. However, clients may still read old versions of blocks from their local caches. Although this drawback is acceptable for some applications, it might not be desirable in general and can be eliminated by the next algorithm.

Write Share Concurrent (WSC): To overcome the coherency problem inherent in WSS, WSC uses a slight modification of the protocol. Instead of changing the write policy from write-back to write-through when a file is shared, the file caching policy is changed to be non-cachable on the clients. This forces the only version of the file to be kept on the server, which guarantees consistency under any circumstances. This approach loads the server with the burden of handling all shared file accesses. However, as long as the file sharing ratio is not too high this approach is acceptable.

2.2 Remote Memory Variable-Size Client Caches

Besides variations in caching policies as presented in the previous section, another orthogonal direction to explore is the usage of *the network*, i.e. remote memory accesses, to fulfill local cache misses. We investigate the use of remote memory by allowing each client to split its local cache into two distinct regions. One region is used to hold local cache contents, whereas the other region is exported to be used by other clients.

Splitting cache memory into two regions and exporting part of it to other clients reveals two questions. How much memory should be used locally, and which clients are allowed to use the exported regions. Considering the overall performance of the system as the target to be optimized, it can be proven that an optimal solution to this partitioning problem exists [Klauser 1994]. The optimum, i.e. the minimum total number of misses in the whole system, is reached when the derivatives of all clients' miss functions with respect to their cache size are equal.

We have considered this strategy by making two sets of runs over the trace data; during the first set the optimal cache partitioning for various global cache sizes has been collected. The second set of runs uses these optimal cache partitions during its operation. In a production environment this two stage process needs to be replaced by a one stage process that uses an on-line cache partition prediction algorithm.

3 Simulation and Trace Data

Proteus [Brewer, Delloracas, et at. 1991], a public domain parallel architecture simulator has been used to evaluate the presented caching policies. Proteus is an execution-driven simulator for parallel architectures. It handles the simulation of both the processing hardware as well as the communication subsystem and the network. It also provides basic operating system services on processing nodes like multithreading, synchronization and communication. The simulation is performed on a processor clock-cycle basis.

Simulated models are fed with the Sprite File System trace data from the University of California at Berkeley [Baker, Hartman, et al. 1991]. These traces are publically available. They contain a variety of different events like read, write, open, close, delete, lookup and many others. For this simulation only read and write events have been used. The traces come from several different file servers each containing data from several 48-hour and 24-hour sampling periods. For this study, however, we only use traces from the main file server. Traces from secondary servers show such a small amount of traffic that the caches usually did not warm up before the simulation was completed. We use three sets of traces, each representing 48 hours of continuous workload. [Tab. 1] shows the actual workload presented to the simulator after some data reduction to eliminate kernel, backup, and trace gathering references from the traces. The first 24 hours of every trace are used to warm up the caches, and only the second 24 hours are counted towards the results.

Although trace data events are tagged with a time stamp in the traces, it is not used in this case. The events are fed into the simulator as fast as the simulated caching model is able to handle them. There are two restrictions that bound the event flow into the simulator. On one hand obviously serial accesses to each file are serialized in the inbound data stream, i.e. the second request is held back until the first one has completed. On the other hand a static limit of maximal 8 outstanding requests from any node is enforced. This limit simulates the maximum number of processes on a node that will perform I/O requests to the server concurrently. This, however, does not restrict the number of processes that run on any node but is a way to control the mean I/O activity of a client.

Data	read				write				read + write	
Set	count	vol.	c %	v %	count	vol.	c %	v %	count	vol.
1.1	223199	898.2	68.6	54.7	101942	744.6	31.4	45.3	325141	1642.8
1.2	61755	254.6	89.2	89.5	7458	29.8	10.8	10.5	69213	284.4
2.1	185279	746.7	58.7	56.7	130535	570.0	41.3	43.3	315814	1316.7
2.2	42508	165.5	62.4	59.1	25607	114.7	37.6	40.9	68115	280.2
3.1	163955	668.2	70.2	59.5	69646	455.4	29.8	40.5	233601	1123.6
3.2	106741	426.0	67.9	59.9	50395	285.6	32.1	40.1	157136	711.6

Table 1: Results of read/write event data reduction. Split-up of number of read and write events and volume of data (in MByte) actually used for each of the six data sets. The table contains the number of events processed and the volume of traffic transferred for both, read and write events. Also listed is the relative percentage of read and write events for both, event count and traffic volume.

4 Results and Discussion

This section compares hit rates acquired for different caching models throughout the network at the same boundary conditions such as size of the caches, network throughput and latency, disk throughput and seek time.

4.1 Disk Access Traffic

All five client cache models LOD, NOC, WTA, WSS, WSC are simulated with cache sizes ranging from 16 to 8192 blocks per client (i.e. 64 kBytes to 32 MBytes). The hit rate behavior of all five models, though, shows only a difference of some percent from each other, especially as caches get large enough to hold a relevant part of the clients' working sets. Using a server cache as a second-level cache reduces these differences still further. This result gives confidence that neither very sophisticated nor very simple coherency schemes do change much in the access traffic to the server disk. The dominant parameter for this traffic is the size of the caches and not the coherency protocol used. Simple coherency protocols can eventually compensate for disadvantages due to inefficient traffic characteristics, by using less space for the cache state data, thus leaving more space to allocate to actual cache data buffers.

4.2 Network Load

Comparing network traffic shows a completely different situation. Here we see more sophisticated schemes substantially reducing traffic on the network. However, it highly depends on type, topology, and speed of the network whether these effects are of any severity in the perspective of the whole system. Especially on the simulated network model, long term utilization of the communication channels is very low. However, request bursts usually found in file access traffic can lead to significant network loads for short periods of time. As networks get faster more rapidly than disks, the importance of network load will even shrink further. The limiting factor in our simulation was more or less the I/O performance of the disk, which was assumed to be several times lower than the network performance. These observations lead to the insight that, under the aspect of a well balanced system, disks can be off-loaded by putting load on the network. Using remote memory accesses to maintain a system-wide distributed cache can help to increase client cache hit rates and thus off-load both the server and its disks. Additionally, the point-to-point network structure handles request bursts more gracefully as the increased load is distributed over large parts of the system, instead of being concentrated onto a single shared communication link.

4.3 Server Cache vs. Fixed-Size Client Caches

Server caching on its own is a very simple way to approach the situation. Nevertheless, we found that it is by far better than any of the client caching schemes under consideration, even when using the most optimistic assumptions about additional coherency traffic (LOD). This effect even grows drastically with increasing cache sizes in the system. Large client caches, as used sometimes now and certainly used more often in the future, perform several times worse than server caches with the same total number of cache blocks. This behavior is depicted in [Fig. 1] for data set 2.

The reason for this unpleasant behavior can be explained by the access patterns of the trace data. The traces hold requests coming from a large number of different workstations arriving at the server. Although the number of different sources has been reduced in the simulation to fold the traces onto the simulated topology, it still shows an unbalanced static and even more unbalanced dynamic usage pattern. This leads to the effect that some clients user their caches efficiently, while others completely underutilize their caches. Still others are far from optimal in their cache hit rates because they have to handle much larger working sets. [Fig. 2] shows the great variety of client cache behaviors with varying cache size. Adding more memory to all client caches only shows significant effects on overutilized caches, while underutilized ones can not make effective use of additional memory. Hence, from a certain point on, adding more memory is only of marginal benefit for the overall performance of the whole system. Unfortunately it is not predictable in advance how much cache memory each client can use efficiently. Moreover, this cache usage pattern does not stay constant over time.

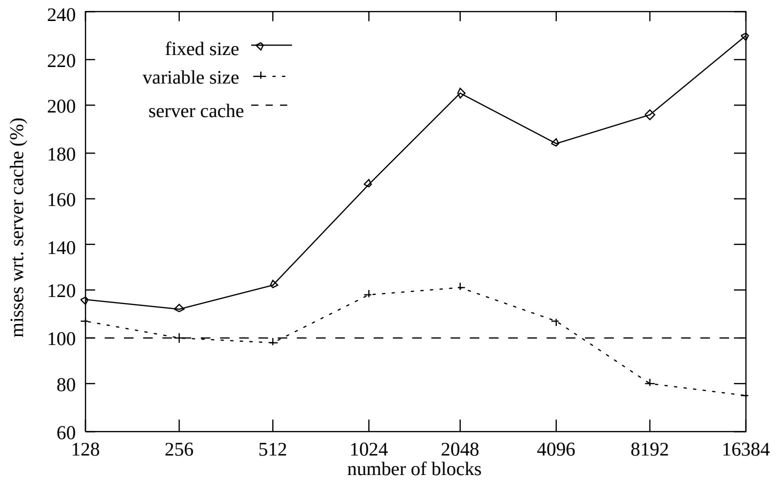

Figure 1: Fixed and variable-size client cache behavior with respect to server cache for data set 2.

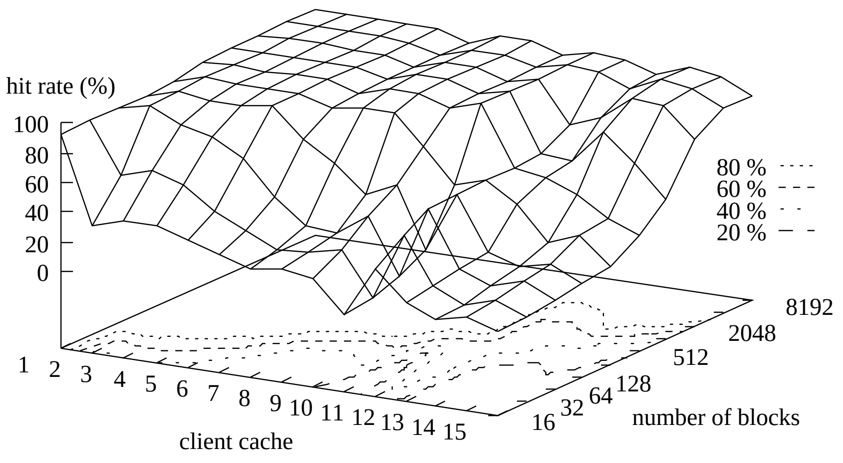

Figure 2: Influence of the cache size on the hit rates of client caches.

4.4 Remote Memory Variable-Size Client Caches

The previous observations lead to the insight that fixed-size caches do not contribute to efficient client caching schemes. Hence, cache memory has to be shared and balanced between clients. By allowing remote memory caches, as described in [Section 2.2], client caching can be made much more efficient.

Simulation studies with varying cache sizes show that a distributed variable-size client cache scheme exhibits almost the same miss rate as a single big server cache. [Fig. 1] shows that distributed variable-size client caches perform very competitively compared to a single big server cache. Due to the increased reference locality in a private client cache compared to a shared server cache, distributed variable-size client caches even perform better than a big server cache for some cache sizes and input data sets.

Although this scheme reintroduces network traffic due to the nonlocal use of memory, it still produces less network traffic than a pure centralized server cache. It frees the server from the burden of processing every single request in the whole system and shows much better hit rates than any of the fixed-size client caching approaches discussed before. Moreover, the additional traffic is distributed throughout the whole system and does not introduce new hot spot communication or processing bottlenecks.

4.5 Network and I/O Speed

To be more confident about the simulated computer model some sets of simulations also have been repeated with varying network and I/O bandwidth parameters. Variations of the network bandwidth ranged from 1 to 200 MBytes/s and variations of the disk throughput ranged from 1 to 10 MBytes/s. All these simulations show that the miss rates reported in the system are fairly independent of these parameters, with variations being in the range of the accuracy of the simulation, which is predicted as $\pm$ 0.5 % by statistical considerations.

5 Related Work

Optimal partitioning of memory for concurrent operations based on multiple unrelated input streams has been investigated by [Thiebaut, Stone, Wolf 1992], [Stone, Turek, Wolf 1992], and [Ghanem 1975]. Our approach of deriving optimal sizes for splitting client cache memory into local and remote parts has some similarities to their work.

[Mohindra, Ramachandran 1991] and [Zhou, Stumm, Li, Wortman 1990] investigate the use of distributed shared memory (DSM) in networks of workstations. Local/remote cache splitting is based on some of these DSM ideas.

[Nelson, Welch, Ousterhout 1988] describe caching approaches taken in the Sprite network file system. Client caches use delayed write-back to reduce server load and vulnerability to crashes.

[Dahlin, Wang, et al. 1994] and [Dahlin, Mather, Wang, et al. 1994] explore the use of remote client memory to improve file system performance in xFS. The approach is based on modifications to the AFS file system to allow direct client-to-client interaction. Their mechanism is based on caching whole files and uses a coherency scheme of write-after-close. Trace driven simulations use the Berkely Sprite and Auspex file system traces.

In the work of [Mann, Birell, et al. 1994] caching strategies in the Echo distributed file system are presented. Client caching with delayed write-back is used to reduce client write traffic to the server. Ordering constraints on write-back allow coherency to be maintained in the case of unreliable clients.

[Biswas et al. 1994] use non-volatile write caches, together with volatile read caches to provide reliability in the case of distributed file systems with client caching. They use synthetic workloads with a commercial production I/O profile.

6 Conclusions

We have compared a number of client caching schemes for high performance networks of workstations and MPPs. Different cache coherency approaches and distribution schemes have been used. Cache hit rates have been compared using centralized server caching as a major reference point.

Variations in miss rate for different coherency schemes used by fixed-size client caches have been found to be negligible compared to the difference between fixed-size schemes and a centralized server cache. To achieve miss rates in the range of a single big server cache it is important to give clients access to remote memory resources. Moreover, clients can adjust their cache sizes among each other for the overall number of misses to reach a minimum. This can be achieved by allowing clients with overutilized caches to use part of the underutilized clients' cache memory.

By increasing networking traffic to access remote parts of client caches, distributed variable-size client caches reduce disk access traffic and thus reach better balanced system resource usage. This approach has advantages over both, centralized server caches and fixed-size client caches.

References

[Baker, Hartman, et al. 1991] Baker, M. G., Hartman, J. H., Kupfer, M. D., Shirriff, K. W., Ousterhout, J. K.: "Measurements of a Distributed File System"; Technical report, University of California at Berkeley, Computer Science Division, July 1991, also appeared in Proceedings of the 13th Symposium on Operating Systems Principles, Oct. 1991.

[Biswas et al. 1994] Biswas, P., Ramakrishnan, K. K., Towsley, D., Krishna, C. M.: "Performance Benefits of Non-Volatile Caches in Distributed File Systems"; Concurrency—Practice and Experience, 6, 4 (1994), 289–323.

[Brewer, Delloracas, et at. 1991] Brewer, E. A., Dellarocas, C. N., Colbrool, A., Weihl, W. E.: "Proteus: A High-Performance Parallel-Architecture Simulator"; Technical report MIT/LCS/TR-516, Massachusetts Institute of Technology, Laboratory for Computer Science, September 1991.

[Coulouris, Dollimore 1988] Coulouris, G. F., Dollimore, J.: "Distributed Systems: Concepts and Design"; Addison-Wessley 1988, ISBN 0-201-18059-6.

[Dahlin, Mather, Wang, et al. 1994] Dahlin, M. D., Mather, C. J., Wang, R. Y., Anderson, T. E., Patterson, D. A.: "A Quantitative Analysis of Cache Policies for Scalable Network File Systems"; Proceedings of the ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, May 1994.

[Dahlin, Wang, et al. 1994] Dahlin, M. D., Wang, R. Y., Anderson, T. E., Patterson, D. A.: "Cooperative Caching: Using Remote Memory to Improve File System Performance"; Proceedings of the Operating Systems: Design and Implementation Conference, November 1994.

[Ghanem 1975] Ghanem, M. Z.: "Dynamic Partitioning of the Main Memory Using the Working Set Concept"; IBM Journal of Research and Development, 19, 9 (1975), 445–450.

[Klauser 1994] Klauser, A. W.: "A Simulation Study for Distributed File Caching in High-Performance Parallel Architectures"; Master's thesis, Graz University of Technology, Austria, Department for Applied Information Processing, January 1994.

[Maffeis 1992] Maffeis, S.: “Cache Management Algorithms for Flexible Filesystems”; Technical report, Institut für Informatik der Universität Zürich (IFI), December 1992.

[Maffeis, Cap 1992] Maffeis, S., Cap, C. H.: “Replication Heuristics and Polling Algorithms for Object Replication and a Replicating File Transfer Protocol”; Technical Report IFI TR 92.06, Institut für Informatik der Universität Zürich (IFI), July 1992.

[Mann, Birell, et al. 1994] Mann, T., Birrell, A., Hisgen, A., Jerian, C., Swart, G.: “A Coherent Distributed File Cache with Directory Write-Behind”; ACM Transactions on Computer Systems, 12, 2 (1994), 123–164.

[Mohindra, Ramachandran 1991] Mohindra, A., Ramachandran, U.: “A Survey of Distributed Shared Memory in Loosely-coupled Systems”; Technical Report GIT-CC-91/01, College of Computing, Georgia Institute of Technology, January 1991.

[Nelson, Welch, Ousterhout 1988] Nelson, M. N., Welch, B. B., Ousterhout, J. K.: “Caching in the Sprite Network File System”; ACM Transactions on Computer Systems, 6, 1 (1988), 134–154.

[Stone, Turek, Wolf 1992] Stone, H. S, Turek, J., Wolf, J. L.: “Optimal Partitioning of Cache Memory”; IEEE Transactions on Computers, 41, 9 (1992), 1054–1068.

[Thiebaut, Stone, Wolf 1992] Thiebaut, D., Stone, H. S., Wolf, J. L.: “Improving Disk Cache Hit-Ratios Through Cache Partitioning”; IEEE Transactions on Computers, 41, 6 (1992), 665–676.

[Zhou, Stumm, Li, Wortman 1990] Zhou, S., Stumm, M., Li, K., Wortman, D.: “Heterogeneous Distributed Shared Memory”; Technical Report CSRI-244, Computer Systems Research Institute, University of Toronto, September 1990.

Appendix A

[Tab. 2] contains hit rates of the various caches for trace data set 2. Abbreviations *CC* and *SC* stand for *client cache* and *server cache* respectively, and *OV* stands for *overall*, which compares the number of misses of the last cache level to the total number of accesses processed in the system. (Entries of * represent the cases where no accesses have been detected, thus no hit rate can be given.)

Data Set 2			SCO	LOD	NOC	WTA	WSS	WSC
32 blocks	CC	read	-	33.4	33.5	33.9	33.5	30.2
		write	-	8.1	8.1	7.5	8.1	8.1
		r+w	-	23.2	23.2	23.2	23.2	21.3
	SC	read	23.2	-	2.1	0.8	2.2	6.5
		write	2.2	-	0.3	2.6	0.5	0.4
		r+w	14.7	-	1.2	1.7	1.4	3.6
	OV	read	23.2	33.4	34.7	34.3	34.8	34.6
		write	2.2	8.1	5.5	2.6	5.6	5.7
		r+w	14.7	23.2	22.9	21.5	22.9	22.9
512 blocks	CC	read	-	73.8	73.8	74.0	73.8	69.3
		write	-	70.5	70.5	70.7	70.4	70.3
		r+w	-	72.4	72.5	72.7	72.4	69.7
	SC	read	45.3	-	14.3	0.6	14.0	26.1
		write	26.6	-	2.2	34.8	2.8	2.6
		r+w	37.7	-	8.1	25.3	8.2	14.9
	OV	read	45.3	73.8	77.5	74.1	77.5	77.3
		write	26.6	70.5	59.5	34.8	60.0	60.0
		r+w	37.7	72.4	70.2	58.2	70.4	70.3
8192 blocks	CC	read	-	98.7	98.7	98.7	98.7	94.4
		write	-	92.2	92.2	92.2	92.2	92.0
		r+w	-	96.1	96.1	96.1	96.1	93.5
	SC	read	85.0	-	32.0	19.2	32.0	84.2
		write	87.4	-	*	78.8	100.0	87.0
		r+w	85.9	-	32.0	77.7	37.2	84.3
	OV	read	85.0	98.7	99.1	98.9	99.1	99.1
		write	87.4	92.2	100.0	78.8	100.0	100.0
		r+w	85.9	96.1	99.5	90.8	99.5	99.5

Table 2: Cache hit rates for data set 2.