This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

Memory Hierarchy Characterization of NoSQL Applications through Full-system Simulation

Adrian Colaso, Pablo Prieto, Jose A. Herrero, Pablo Abad, Lucia G. Menezo, Valentin Puente, Jose A. Gregorio

***Abstract*—In this work, we conduct a detailed memory characterization of a representative set of modern data-management software (Cassandra, MongoDB, OrientDB and Redis) running an illustrative NoSQL benchmark suite (YCSB). These applications are widely popular NoSQL databases with different data models and features such as in-memory storage. We compare how these data-serving applications behave with respect to other well-known benchmarks, such as SPEC CPU2006, PARSEC and NAS Parallel Benchmark. The methodology employed for evaluation relies on state-of-the-art full-system simulation tools, such as gem5. This allows us to explore configurations unattainable using performance monitoring units in actual hardware, being able to characterize memory properties. The results obtained suggest that NoSQL application behavior is not dissimilar to conventional workloads. Therefore, some of the optimizations present in state-of-the-art hardware might have a direct benefit. Nevertheless, there are some common aspects that are distinctive of conventional benchmarks that might be sufficiently relevant to be considered in architectural design. Strikingly, we also found that most database engines, independently of aspects such as workload or database size, exhibit highly uniform behavior. Finally, we show that different data-base engines make highly distinctive demands on the memory hierarchy, some being more stringent than others.**

***Index Terms*—Memory Hierarchy, Big-data, NoSQL, Cache Hierarchy, Benchmark characterization.**

I. Introduction

CONCERNING Information Technologies, one of the broad fields with a large social and economic impact is Big-data Analytics. This term includes a wide technology spectrum with increasing influence in such different areas as Biology, Economics or Healthcare. In these heterogeneous scenarios, the most relevant feature from the computing perspective depends on the nature of the data analyzed. Compared to conventional applications, Big-data applications need to handle an unprecedented volume of and highly heterogeneous data (5Vs model, volume, velocity, variety, value and veracity). In recent years, software has rapidly evolved to provide support for Big-data environments. Centralized storage and processing, able to manage data analytics through relational models and SQL querying, are gradually being replaced by alternatives more focused on the software side to cope with the need for scalability under constrained cost. These approaches are based on fully distributed storage and processing frameworks running with commodity hardware [1]. Alternative data models (defined as NoSQL (Not only SQL) databases), such as those based on columns [2], graphs [3] or documents [4] seem to be much more appropriate in this environment [5][6]. Additionally, the adoption of mechanisms which are able to meet data processing speed demands becomes essential, with solutions such as in-memory storage [7] or high-performance processing frameworks [8].

In contrast to this software-centric paradigm shift, the underlying hardware has, in general, little to no specialization. In most cases, these complex software stacks run on top of computer clusters, made up of low-cost commodity hardware. For this reason, there is great interest in evaluating how this HW-SW distinctiveness can affect system behavior and performance. Numerous previous works [9][10][11][12] have made a remarkable effort in the characterization of Big-data applications. These works make use of benchmark suites covering broad application scenarios, in most cases running on current hardware. Therefore, they rely on hardware performance monitoring units for characterization. That approach has enabled the detection of some relevant mismatches between the demands of Big-data workloads and today's processor micro-architecture [9][13].

Despite the relevant findings of all these characterization works, the methodology employed has a significant limitation, which is the fixed nature of the microarchitecture under study. This limitation leaves many questions unanswered such as: what is the appropriate size for instruction caching? and for data? Are these applications responsive to performance mechanisms such as replacement policy or prefetching? What is the sharing degree in these multi-threaded applications?

In this paper, we conduct the experiments required to provide answers to these questions making use of an alternative methodology: using a full system simulation tool capable of (1) allowing the large software stack to be executed without changes and, (2) being fast enough in performing the complex warmup of the applications in a feasible amount of time. As

Submission Date:

This work was supported in part by the Spanish Government (Secretaría de Estado de Investigación, Desarrollo e Innovación) under Grants TIN2015-66979-R and TIN2016-80512-R.

All the authors are with the Computer Engineering Group, University of Cantabria, Santander, 39005 Cantabria, Spain (e-mail: [colasoa, prietop, herreroja, abadp, gregoriol, vpuente, monaster]@unican.es).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

the core of our methodology we use gem5 [14] which provides an adequate framework to achieve such goals. Beyond the full-system simulation framework, the YCSB benchmarking tool [15] has been used to characterize in detail a representative set of NoSQL databases: Cassandra [16], MongoDB [17], OrientDB [18] and Redis [19]. Our results confirm some of the previous findings about this kind of applications (such as the large instruction working-set [13][9][20]) and, thanks to the simulation framework, provide a much more detailed description of the cache-hierarchy implications:

- We observe homogeneous behavior in many aspects of these applications. Despite their differences, the four databases present a similar fraction of load/store operations (extremely homogeneous in the case of stores), as well as similar data working sets.
- In general, we found that NoSQL applications present cache-friendly behavior, similarly to conventional benchmark suites. The evaluated workloads benefit from an increased cache size.
- We are able to quantify the instruction cache capacity required by NoSQL applications in order to reduce MPKI levels to those observed in conventional benchmarks.
- We evaluate the effect of alternative multi-level cache configurations on memory performance, showing the benefits of improving the L1 capacity in contrast to the small benefit of increased LLC size.
- We observe that LLC replacement policies are critical for this kind of applications, in contrast to the results obtained for conventional workloads.
- Finally, we have also been able to provide detailed information about different prefetching policies and the sharing degree of instruction and data blocks in the LLC.

The rest of the paper is organized as follows: section II describes the YCSB framework, NoSQL databases and the simulation tools. Section III defines all the aspects concerning the evaluation methodology. Sections IV and V present our results and characterization. Finally, we describe some related work and summarize our main conclusions in sections VI and VII respectively.

II. Simulation Stack

In this section, we describe the framework employed for characterization, including both the software and hardware stacks.

A. YCSB FRAMEWORK

In order to run all NoSQL applications under fair conditions we make use of the Yahoo! Cloud Serving Benchmark (YCSB) [15]. This framework has been designed to assist in the evaluation of different cloud serving systems and is intensively employed for database performance evaluation and comparison (more than 350 citations according to the ACM Digital Library). Additionally, some individual workloads of this benchmark are currently being used as part of alternative benchmarks, such as SPEC Cloud IaaS 2016 [21] (focused on measuring the performance of cloud implementations) and CloudSuite [9] (benchmarking dominant scale-out workloads). The YCSB architecture consists of a synthetic generator for database operations and multiple interfaces for different commercial software. The *Workload Generation Client* is responsible for the generation of the data to be loaded into the database. It is also in charge of the later definition of the operation-mix that characterizes each workload. The properties defining these workloads include aspects such as read/write mix and distribution (latest, uniform, etc.) of the database records as well as size and number of fields in each record. The core package includes a set of pre-defined workloads that tries to model different applications (such as picture tagging, user status updates or threaded conversations).

The *Database Interface Layer* is the module in charge of translating simple requests from client threads into calls to specific databases. Currently, this component implements interfaces for multiple software suites (more than 25 according to the *github* repository), covering the different data models of NoSQL applications (column-group, document, graph) and multiple existing read/write performance optimizations.

B. NoSQL AND CONVENTIONAL APPLICATIONS

To make characterization feasible, we limit our evaluation to four of the wide range of available applications. The ones selected try to cover multiple features present in today's NoSQL data serving, such as alternative data models (column, graph and document-based) and performance optimizations (distributed disk and in-memory storage). To provide a reference point, we evaluate under the same conditions conventional benchmarks such as: SPEC CPU2006 [22], PARSEC [23] and NPB [24]. Next, a brief description of all the characterized software is provided.

1) Cassandra (NoSQL database, column oriented)

Apache Cassandra [16] is currently one of the most popular wide-column store databases [25]. This distributed data management system has been designed to work with large data volumes on top of commodity hardware, providing high availability and fault tolerance features. With a completely decentralized operation, data is distributed throughout the cluster, there is no master node and every node has a homogeneous role. Fault-tolerance is implemented through the automatic replication of data into multiple nodes. Cassandra makes use of its own query language (Cassandra Query Language or CQL) and also has Hadoop [1] integration (MapReduce support). Nowadays, more than 600 companies employ Cassandra software, with major users such as Microsoft, IBM, Facebook or Apple [26].

2) MongoDB (document-oriented data model)

MongoDB [17] is a document-oriented database software designed to provide scalability, performance and high availability. Documents are stored as binary JSON objects (schemas), supporting field and range queries as well as regular expression searches. Data distribution across multiple machines is implemented through *sharding*. In this way, it

supports very large data sets at high throughput rate. High availability and fault-tolerance is implemented through replica sets. Writes and reads are done on the primary replica while secondary replicas maintain a copy of the data. When a primary replica fails, the replica set automatically conducts a selection process to determine which secondary should become the primary. As in the case of Cassandra, MongoDB is also one of the most popular document stores, with a large diversity of users [25].

3) Redis (In-memory Store)

Redis [19] is an in-memory data structure store, used as a database, cache and message broker. It supports a large number of different data structures (strings, hashes, lists, sets, bitmaps…), it has built-in replication, different levels of on-disk persistence and it provides high availability. Automatic data partitioning is performed across Redis nodes, and master-slave replication is implemented and employed for both scalability and data redundancy. A list of well-known companies using Redis includes Twitter, Github, StackOverflow, Pinterest, among many others.

4) OrientDB (graph data model)

OrientDB [18] is a multi-model database where relationships are managed as graph databases, with direct connections between records. It supports schema-less, schema-full and schema-mixed modes. OrientDB also has a master-less distributed architecture where each server can read and write, allowing horizontal scale-up without bottlenecks. Typical SQL querying language can be used by simply adding the necessary extensions to enable graph functionality. According to the DB-Engines graph database ranking [25], OrientDB is one the most popular graph databases.

5) SPEC, PARSEC & NPB (Conventional Benchmarking)

SPEC CPU2006 [22], PARSEC [23] and NAS Parallel Benchmark [24] are three of the most significant benchmark suites in computer architecture research. The characterization of these suites has been addressed by previous research works [27][23] and their workloads are widely employed for micro-architecture research and evaluation. In many Big-data characterization works both PARSEC and SPEC benchmarks have usually been employed as a reference [9][13][28][11], highlighting the main differences between emerging scale-out applications and these suites. Following a similar approach, in this work we will also include the three aforementioned suites in the characterization process. SPEC CPU2006 is an industry-standardized suite of serial programs (not intended for parallel machine evaluation) designed to stress both the processor and memory subsystem. In contrast, PARSEC and NPB focus on the evaluation of parallel machines.

C. HARDWARE (Simulation Framework)

For the proposed characterization we make use of the gem5 simulation framework [14]. Gem5 provides detailed CPU and memory system models, as well as supporting most commercial ISAs. The Gem5 simulator is broadly available (BSD license) through a publicly accessible source repository, constantly updated by the gem5 community. Our framework version has been forked from an up-to-date repository commit. We have fixed/implemented the necessary features to conduct the experiments for all the workloads under consideration. Original code has been modified to support an Ethernet network interface, switch and the corresponding links. Thus, we can simulate a whole cluster and hence support multi-node environments, which is the common case in NoSQL-based applications. The framework supports the simulation of any number of nodes (each of them with its own configuration) connected through Ethernet devices.

Our framework also includes virtual machine (VM) based simulation acceleration [29]. During boot and warmup phases, the VM will run a replica of the simulated machine at native speed. The VM and simulated system state are synchronized at coarse time intervals. The support for multithreaded event queues in gem5 allows each CPU to be run in the simulated system in a separate process in the VM. This enables near native speed even in multinode simulated systems. Using this approach, the applications are “fast-forwarded” to their interest point in a reasonable amount of time. In the case of NoSQL applications, where large databases must be generated prior to evaluation, the fastest non-vm-assisted simulation mode (i.e. atomic) would require weeks or even months of simulation to reach the region of interest, which would make the study unfeasible.

Besides the mentioned modifications, the checkpointing, simulation and profiling processes have been completely automated through scripting. The automation eases the characterization process, where the generation of a large number of results is required. The source code of the simulation framework, prepared to run NoSQL applications, has been made publicly available through an online repository at: https://github.com/abadp/gem5-NoSQL.

III. METHODOLOGY

A. Software Stack Configuration (YCSB & NoSQL Apps)

Every application evaluated runs on top of a GNU/Linux OS (Debian 8 using gem5 compatible kernel v3.18.34). YCSB and NoSQL applications make use of Oracle’s open source Java Virtual Machine, 1.7 version (python, 2.7.9 Version). We have employed up-to-date software versions for all the NoSQL applications: Cassandra-2.2.5, *MongoDB*-3.2.11, *Redis*-2.8.17, *OrientDB*-2.1.2 (community edition). Concerning conventional benchmarks, the latest version available is used in all cases (SPEC CPU2006 V1.2, PARSEC 3.0 and NPB3.3).

Client and database configuration are two key aspects for the correct characterization of NoSQL workloads. As mentioned before, the content of the database has been generated through the YCSB client. The number of records required to load the database is calculated so that it has an overall size of 1GB. This database size has been chosen to fit into the 4GB Main Memory of the simulated machines (see section C). This is the recommended configuration in scale-out applications for optimal database performance [11]. Every NoSQL engine is configured making use of the default parameters described in YCSB repository. Before evaluation, each NoSQL workload

was fine-tuned to provide the maximum performance. Making use of the simulation framework (gem5), we perform a whole run of all workloads for each application, gradually increasing the number of client threads. Throughput results provided by YCSB are collected, identifying the number of threads that maximizes performance. This is the value selected for the whole characterization process.

B. Workload Generation

Simulation-based methodologies are usually limited by the execution time overhead caused by detailed hardware simulation. To achieve a feasible characterization, we only use accurate hardware simulation during the execution of a significant fraction of the Region of Interest (ROI) of each application. On reaching the interest point with VM acceleration, a checkpoint (which includes all architectural cluster state, i.e. processor, memory, network, etc.) is taken. The checkpoint will be loaded subsequently in detailed architectural simulation. Starting from each checkpoint, the memory hierarchy is warmed up for a sufficient number of cycles before starting to collect statistics. In this way we minimize the effect of compulsory (cold) misses and warm-up non-architectural states (prefetchers, replacement policy, etc.).

TABLE 1: CONVENTIONAL WORKLOADS

SPEC CPU2006 INT
astar, bzip2, gcc, gobmk, h264ref, hmmer, libquantum, mcf, omnetpp, sjeng, xalancbmk
10 Billion instructions simulated (500M warmup). Checkpoint taken after the execution of half of the iterations in the main loop**. Input size**: reference.
SPEC CPU2006 FP
GemsFDTD, bwaves, cactusADM, calculix, dealII, games, gromacs, lbm, leslie3d, milc, namd, povray, soplex, sphinx3, wrf, zeusmp
10 Billion instructions simulated (500M warmup). Checkpoint taken after the execution of half of the iterations in the main loop**. Input size**: reference.
PARSEC 3.0
blackscholes, bodytrack, canneal, dedup, facesim, ferret, fluidanimate, freqmine, raytrace, streamcluster, swaptions, vips, x264
10 Billion instructions simulated (500M warmup). Checkpoint taken at the beginning of annotated ROI. **Input size**: simlarge.
NPB 3.3.1
BT, CG, FT, IS, LU, MG, SP, UA
2 Iterations of the main loop inside ROI simulated (500M cycles warmup). Checkpoint taken at the middle iteration of main loop. **Input Size**: B class.

For multi-core architectures, one thread per simulated core is employed in the case of parallel applications. SPEC applications execute one instance of the same workload per available core. In this special case, we implement a synchronization process that guarantees that each application instance is executing ROI instructions during the characterization process.

YCSB provides a package of standard workloads (See Table 2 for details), with pre-defined operation mixes and access patterns [15]. These workloads are employed for the characterization process. The preparation of YCSB workloads for detailed simulation requires a different process. In this case the definition of a ROI is not required, because the whole workload run is employed. As mentioned previously, database generation is performed through VM-assisted simulation. A checkpoint is taken after the load process is finished. This process might take a few minutes in real hardware. The execution time of detailed simulation can be controlled through YCSB runtime parameters, in this case the total number of operations performed by YCSB clients. For each database, we gradually increase the number of records performed at runtime until reaching a value that requires the execution of approximately 10 Billion instructions, a similar simulation length to the one employed for PARSEC, SPEC and NPB workloads. Table 1 includes a brief description of all the workloads generated for the evaluation process, detailing for each of them the benchmark suite, input size and other relevant information.

TABLE 2: NOSQL WORKLOADS

YCSB						
Database size	~1GByte					
Operation count	Equivalent to 10 Billion Instructions					
Insertion retry limit/ interval	1 / 3 secs.					
WORKLOAD	A	B	C	D	E	F
Field Count / Length	10/100	10/100	10/100	10/100	10/100	10/100
Read proportion	0.5	0.95	1	0.95	--	0.5
Read all fields	true	true	true	true	--	true
Update proportion	0.5	0.05	--	--	--	--
Insert proportion	--	--	--	0.05	0.05	--
Insert order	--	--	--			--
Scan proportion	--	--	--	--	0.95	--
Scan length/distr.	--	--	--	--	100/uni	--
RWM proportion	--	--	--	--	--	0.5
Request Distribution	zipfian	zipfian	zipfian	latest	zipfian	zipfian

C. System & Simulation Configuration

Sections IV and V make use of different configurations for the processor and cache hierarchy. Basic cache features evaluated in Section IV, run in a system architecture with a single core and the cache hierarchy is made up of one level, separating data and instructions. In contrast, for the rest of the characterization process, performed in Section V, we model up two servers with a 4-core CMP with a realistic multi-level cache hierarchy. The first two levels of the cache hierarchy are private for each core, in contrast to the shared Last Level Cache (LLC). In all cases the gem5 implementation of X86-64 is employed. The main parameters associated with this memory hierarchy are summarized in Table 2.

When client-server applications are simulated (NoSQL databases), we model a "clustered" configuration made up of two different nodes connected through Ethernet devices. The client node runs YCSB client instances while the server node is devoted to NoSQL applications. Thus we eliminate the possible software interference that the memory hierarchy could undergo. Statistics are collected only for the server-side node. Given the limited amount of resources and time available for simulation (the simulation of multiple nodes exponentially increases memory footprint and simulation time), we limit our evaluation to single-node databases. Making use of Cassandra, we have compared main simulation

results (instruction mix, cache hierarchy miss rate) for two different configurations, 1-node and 2-node databases, observing in all cases minor differences. Additionally, in order to limit simulation time the appropriate number of operations has also been validated. We have measured the MPKI evolution for a number of operations ranging from 10 to 10,000 (notice that the number of operations in [11] is 100,000). Three different Data Cache sizes (16Kb, 128Kb and 8Mb) were evaluated. The results show that no significant variations are found over 1,000 operations. For the rest of experiments, we apply a minimal 5× margin (5,000 operations) to ensure correctness.

TABLE 3: MULTI-LEVEL HIERARCHY CONFIGURATION

Private Caches	(L1)Size/Associativity / Block Size / Access Time	32KB I/ 32KB D, 8-way, 64B, 1 cycle
	(L2)Size / Associativity / Blck Size / Access Time / Type	256KB Unified, 8-way, 64B, 4 cycles, Exclusive with L1
Shared L3	Size / Associativity / Block Size/Type	8MB, 16-way, 64B, Mostly Inclusive
	NUCA Mapping	Static, interleaved by LSB
	Coherence Protocol	MOESI snooping
	Data Slice Size/Access Time	2MB / 6 cycles
Mem.	Capacity/Access Time /Bandwidth	4GB /240 cycles / 32GB/s

D. Validation

In those cases where the comparison is possible, we use the processor performance counters to validate the simulation results. These tests are run using the client-server model, replicating the database content (generated with the same YCSB commands) and using the same workloads (WA to WF). In our setup, each node makes use of one Intel Xeon X5650 chip running at 2.67 GHz and a main memory of 48GB. We access the Performance Monitoring Unit (PMU) of the processor through the Linux perf tool [30].

IV. APPLICATION CHARACTERIZATION

In this section, we analyze the basic aspects concerning cache hierarchy design. For each workload, the number of executed instructions is profiled and data/instruction cache sensitivity is evaluated. To improve the readability of the results, graphs only include minimum, maximum and average values for conventional benchmarks.

A. Instruction Profile

To understand the potential effect that load/store instructions have on performance, the first step consists of the evaluation of the fraction of memory operations in the instruction mix. Figure 1 represents the number of memory operations per thousand instructions for each workload, distinguishing between Load (LPKI bar) and Store (SPKI bar) instructions.

Focusing on the results of NoSQL applications, the fraction of memory operations seems to remain consistent despite the different access patterns of each workload. Only Workload E, querying short ranges of records (instead of individual records), presents a different fraction of memory operations. Each NoSQL application seems to deal with these short ranges in a contrasting way. In some cases, the number of load/stores grows (*MongoDB*) while in others it decreases (*OrientDB*, *Redis*).

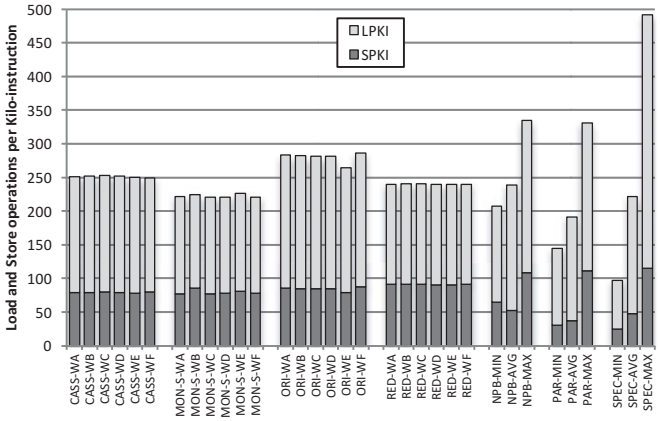

Figure 1. Instruction profile, number of memory operations for each 1000 instructions retired.

Compared to NoSQL applications, conventional benchmarks present much more dissimilar behavior concerning memory operations. From Min to Max bars, the number of load/store instructions can double (PARSEC) or even be 5 times larger (SPEC). Both conventional and NoSQL suites present similar average values for memory operations, close to 25% of total instructions executed. However, SPKI results reveal relevant differences in the number of store operations. NoSQL workloads nearly double the number of store operation compared to the average results of conventional benchmarks. This means that improving Store efficiency could have a larger performance impact in this kind of applications.

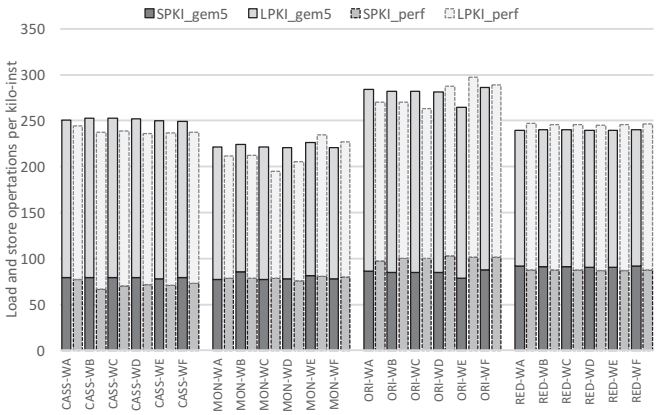

Figure 2. Comparison of Load/Store profile making use of two different characterization methodologies (Simulation & PMU).

This set of results has been validated with the methodology described in section III.D. Running the same YCSB commands and making use of the hardware counters, we re-calculate the fraction of Load and Store operations for each workload. Figure 2 shows the results obtained for validation. Solid bars represent simulated results while dotted bars show the numbers obtained with perf. As can be seen, deviation is minimal (less than 5% on average and a maximum of 12%) and the uniformity of NoSQL workloads is confirmed, as well

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

as the simulated stack consistency.

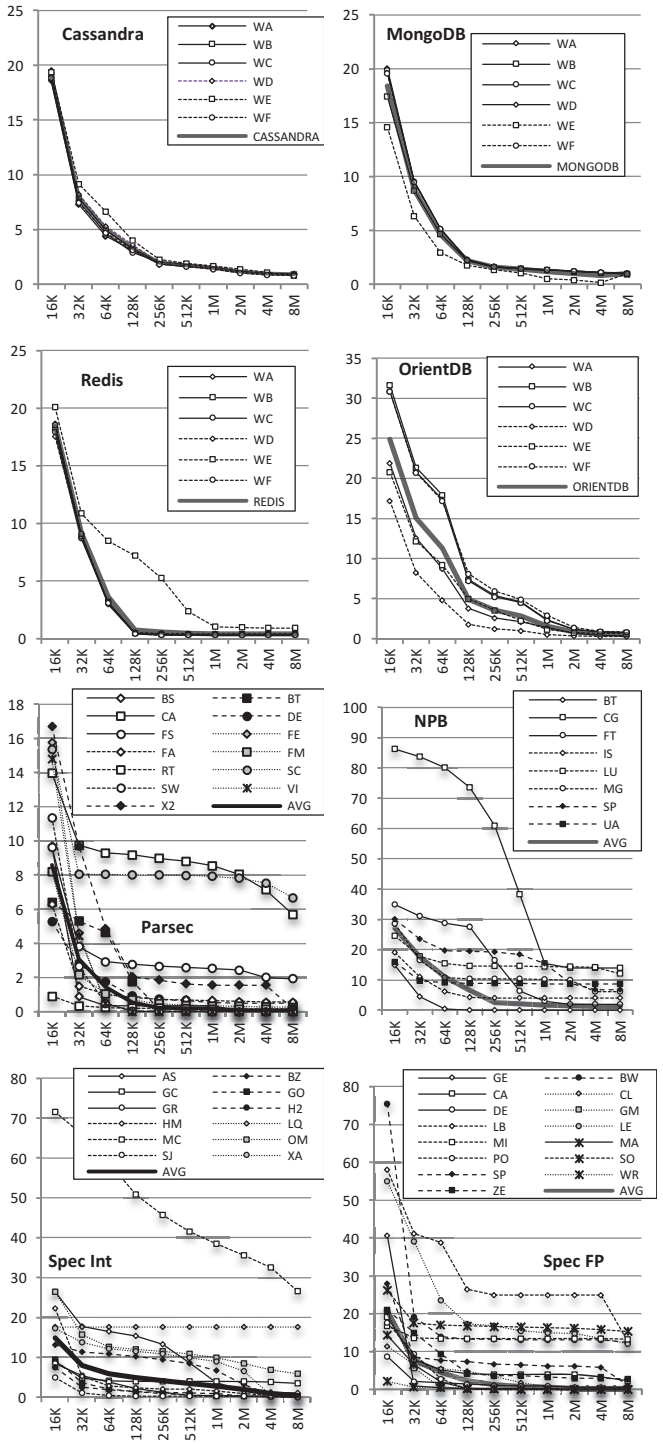

Figure 3. Data working-set for each benchmark. Y-axis represents number of misses for each 1000 instructions retired (MPKI).

B. *Data Working set*

For each workload, we conduct a cache sensitivity analysis through the simulation of multiple cache sizes. We modeled a single-level cache hierarchy ranging from 16KB to 8MB. The 16KB configuration is directly mapped, 32KB cache is 2-way associative, 64KB cache is 4-way associative and so on. Every configuration makes use of 64B blocks and true LRU replacement policy. Figure 3 presents the results obtained from all the simulations performed. The x-axis represents the different cache sizes simulated on a logarithmic scale, whereas the y-axis shows the number of cache misses per one thousand instructions. It should be noted that the y-axis is not constant for every graph in the figure, ranging from 25 (NoSQL benchmarks) to 100 (NPB).

The D-cache sensitivity analysis reveals that, as well as conventional workloads, NoSQL applications exhibit cache friendly behavior (larger cache sizes lead to lower miss rates). Cassandra and MongoDB applications show a highly uniform cache behavior independently of the workload characteristics. The variability increases for *Redis* and *OrientDB*, where some differences arise depending on the workload simulated. OrientDB exhibits the worst data cache performance for those workloads with a significant fraction of read operations. Workloads WB, WC and WF nearly double the mpki results of the rest of applications. These results confirm prior work observations about the larger memory footprint of OrientDB on read operations. Despite this, in all cases the cache miss rate shows smooth exponentially decreasing behavior with increasing cache sizes. Conventional benchmarks exhibit much more dissimilar results, with planar behavior (*streamcluster*, *libquantum* or *LU*) or sudden drops (*x264*, *lbm* or *CG*) not seen in NoSQL results.

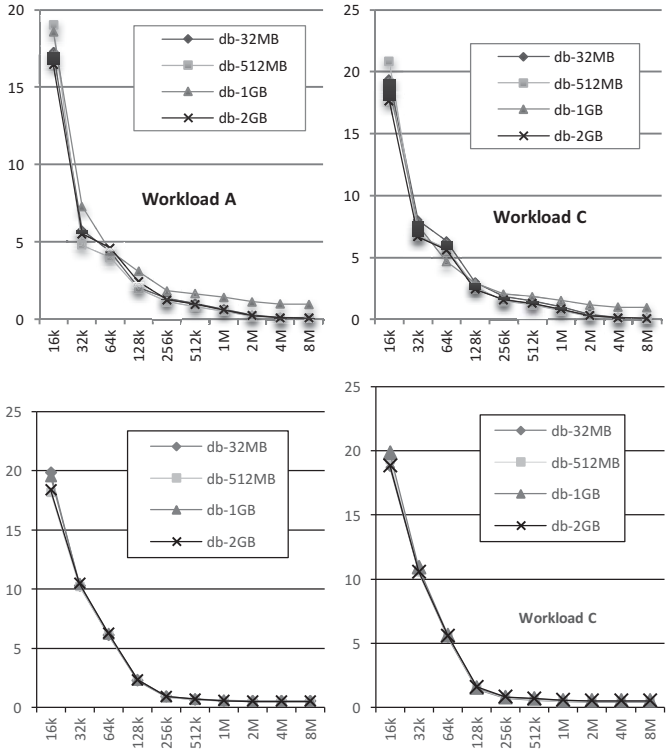

Figure 4. Working-set evolution for different database sizes. Cassandra (above) and MongoDB (below) results.

Absolute MPKI results vary in a much larger range in the case of conventional benchmarks, reaching for some specific applications, initial values close to 100 (mcf or bwaves). However, looking only at average values (geometric mean), we can conclude that both conventional and NoSQL workloads have similar MPKI. Miss results for the smallest cache size range from 10 to 20 for all the benchmarks analyzed. We can also analyze the amount of cache required

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

by each benchmark to reach an average MPKI value below 1, finding the following results: *SpecInt*: 4MB, *SpecFP*: 512KB, *NPB*: >4MB, *Parsec*: 128KB, *Cassandra*: 4MB, *MongoDB*: 4MB, *Redis*: 128KB, *OrientDB*: 2MB. Excluding NPB, with a much larger working set, we can conclude that both conventional and NoSQL benchmarks have a similar data footprint.

C. Data Working-set sensitivity to Database Size

The results obtained in the previous section seem to indicate that the data working-set seen by the cache hierarchy has little relation with the size of the data stored by the database software. To confirm this observation, we carry out an additional experiment analyzing the working-set evolution for different database sizes. Figure 4 shows the results obtained for four different database sizes (32MB, 512MB, 1GB and 2GB) under two applications, *Cassandra* and *MongoDB*. For each workload and size, we repeat the working-set evaluation performed in the previous section, cache size ranging from 16Kbyte to 8Mbyte. As can be seen, few or no differences can be observed in the results obtained. This behavior is consistent for all the NoSQL applications evaluated, and confirms our previous assumption. The working-set observed by the cache hierarchy has no direct relation with the amount of data stored and managed by the NoSQL application.

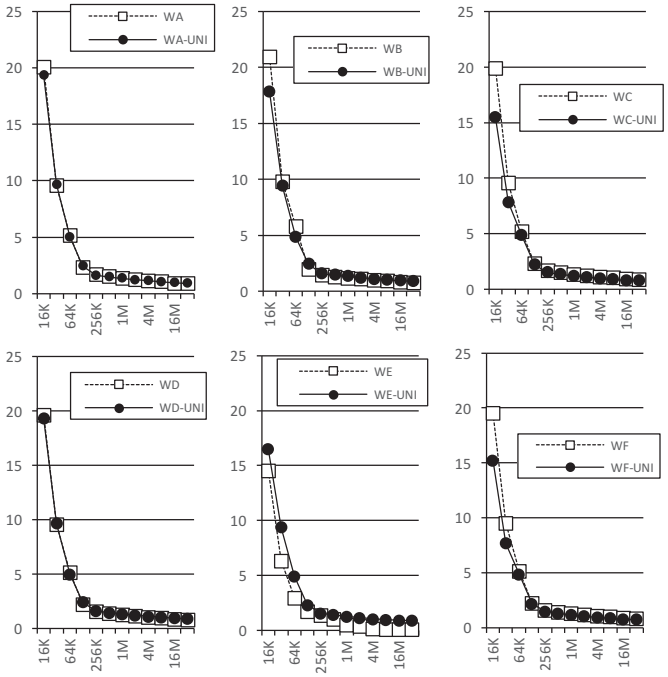

Figure 5. Working-set evolution for Uniform record distribution. *MongoDB* results.

D. Data Working-set Sensitivity to Record Distribution

Most of the core workloads defined by YCSB make use of a record distribution that exploits data locality (latest for WD and zipfian for WA, WB, WC, WE, WF). We repeat a similar experiment to the one performed in previous section to evaluate the effect of the record distribution chosen on data working-set. Figure 5 shows the differences observed when the distribution of each core workload moves from its original value to a uniform one. The application employed is MongoDB. As can be seen, the differences are minimal in most cases, being WE the only workload where uniform records slightly increase the working-set observed.

We conduct a second experiment, making use in this case only workload WA, evaluating all the distributions available in YCSB code. As well as the previously mentioned distributions, we include Hotspot, Sequential and Exponential patterns in the results of Figure 6. We also observe in this case that the record distribution has a minimal effect on the Data working-set of the applications evaluated.

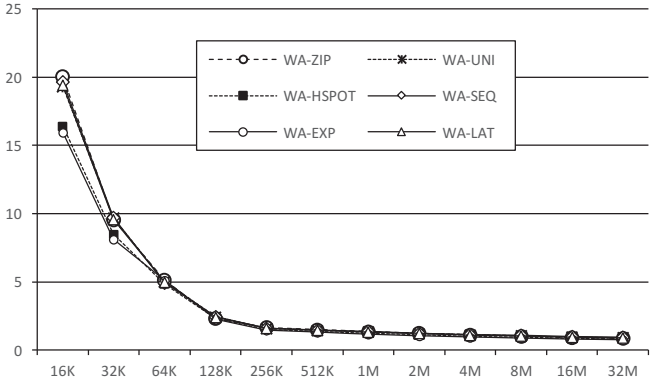

Figure 6. Working-set evolution for different record distributions. *MongoDB* results.

E. Instruction Working set

A separate single cache instruction with sizes ranging from 4KB to 8MB will be used. Direct mapping for smallest size and associativity growing along with size, 64B blocks and LRU replacement. Figure 7 shows the results obtained, again measuring misses per thousand instructions for every workload. In this case, the y-axis is constant for every benchmark analyzed.

The results show that instruction access pattern is cache friendly for Big-data benchmarks. Significant differences can be observed among NoSQL engined, being Redis the application with the largest instruction working-set. The possible cause behind this behavior might be the intensive utilization of TCP-related system calls to access the in-memory data [31]. Concerning variability, NoSQL results seem to be much more uniform than for the rest of benchmarks. The working set values observed for each workload of the same database present minimal differences. This is an expected result, because the different workloads in each NoSQL graph make use of the same software stack (only recording mix changes). In contrast, each line of SPEC, NPB and PARSEC figures corresponds to a different application, with a different instruction footprint.

If we analyze absolute MPKI values, we observe significant differences between NoSQL and conventional suites. NoSQL workloads exhibit a much larger instruction working-set, with similar MPKI values as those observed for data (notice that for small cache sizes instructions exhibit a larger MPKI). This result confirms the findings of previous characterization works [32][33][34][9][13], where the L1 cache size is the main performance bottleneck of current cache hierarchies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

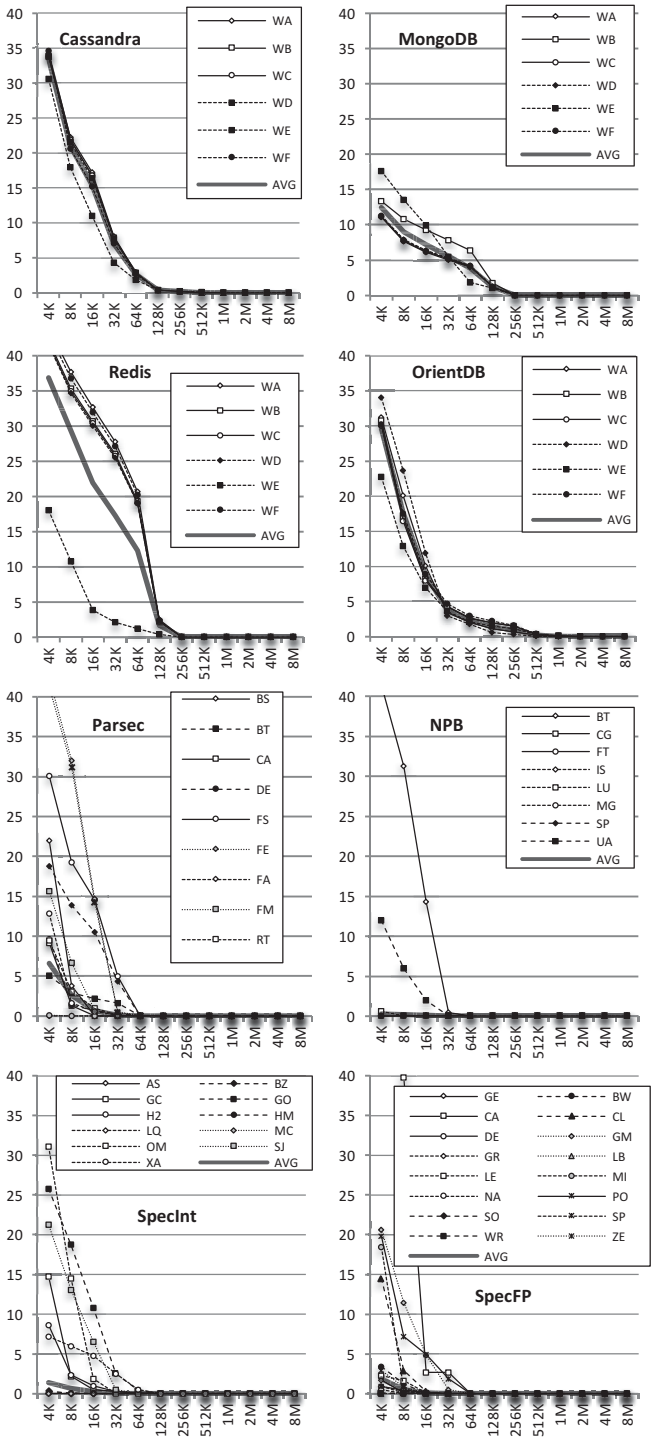

Figure 7. Instruction working-set. Y-axis represents the number of misses for each 1000 instructions retired (MPKI).

V. Multi-level Hierarchy performance

In this section, we make use of a three-level cache hierarchy for evaluation. This configuration attempts to mimic the cache configuration of current commercial processors. The main parameters of this configuration are listed in Table 2. As well as a preliminary MPKI evaluation, we also include results evaluating the sensitivity of NoSQL applications to replacement policy and hardware prefetching. Additionally, we conduct a final experiment measuring the sharing degree of the cache blocks, attempting to understand the possible implications for memory coherence/consistency.

A. MPKI across the Memory Hierarchy

Figure 8 represents the miss rate obtained at each cache level for the configuration in Table 2, where the y-axis represents the number of misses per one thousand instructions. As well as this configuration, labeled BASE in the graphs, we include an additional bar corresponding to a different L1 configuration, so that we can confirm the trend shown in the previous section. With L1x2 we show the results when doubling the L1D and L1I capacity (from 32KB to 64 KB). Moreover, the LLC graph includes a third configuration, labeled L3x2, where the LLC cache size is doubled. The rest of the configuration values remain constant for every simulation. In order to improve graph readability, we only represent min/gmean/max values for the conventional benchmarks and employ the same y-axis scale.

One of the key observations in these results is the high impact of instruction working-set on NoSQL applications. As can be seen, the number of misses in L1I is considerably larger for any of the NoSQL workloads. Additionally, doubling L1I capacity significantly improves MPKI results for NoSQL workloads. In some cases, such as *MongoDB* and *Redis*, the number of misses in first-level instruction cache exceeds those in the data cache. These results confirm the relevance of instruction working-set in these emerging applications, as pointed out in the exploration performed in previous section.

L2 results reveal a side effect caused by increasing L1 capacity. The MPKI values of NoSQL applications improve in L2 when L1 size is doubled, while conventional benchmarks obtain similar values in both cases. Concerning LLC, results show that even at this level NoSQL applications reveal a cache-friendly behavior. Doubling their capacity clearly has a beneficial effect on most of the workloads analyzed, reducing MPKI by 30% in some cases.

B. Replacement Policy

In order to compare the degree of temporal locality of both NoSQL and conventional applications, we evaluate miss-rate in the presence of two different replacement policies. We make use of the well-known LRU policy in contrast to a simple random victim selection (RAND). Figure 9 shows the results obtained for both, the first private (L1D) and the last shared (LLC) cache levels. For each experiment, LRU and RAND policies are applied to the evaluated cache level, while the rest of the cache hierarchy uses LRU policy (e.g. in the LLC evaluation of RAND policy, the L1 and L2 caches take advantage of LRU policy). Graphs show normalized LRU misses. To improve readability we only provide maximum, minimum and geometric mean values for the conventional benchmarks analyzed.

The results obtained for the L1 Data cache seem to indicate that both NoSQL and conventional workloads present a similar degree of temporal locality. Most applications present a similar MPKI improvement in this level. Cassandra seems

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

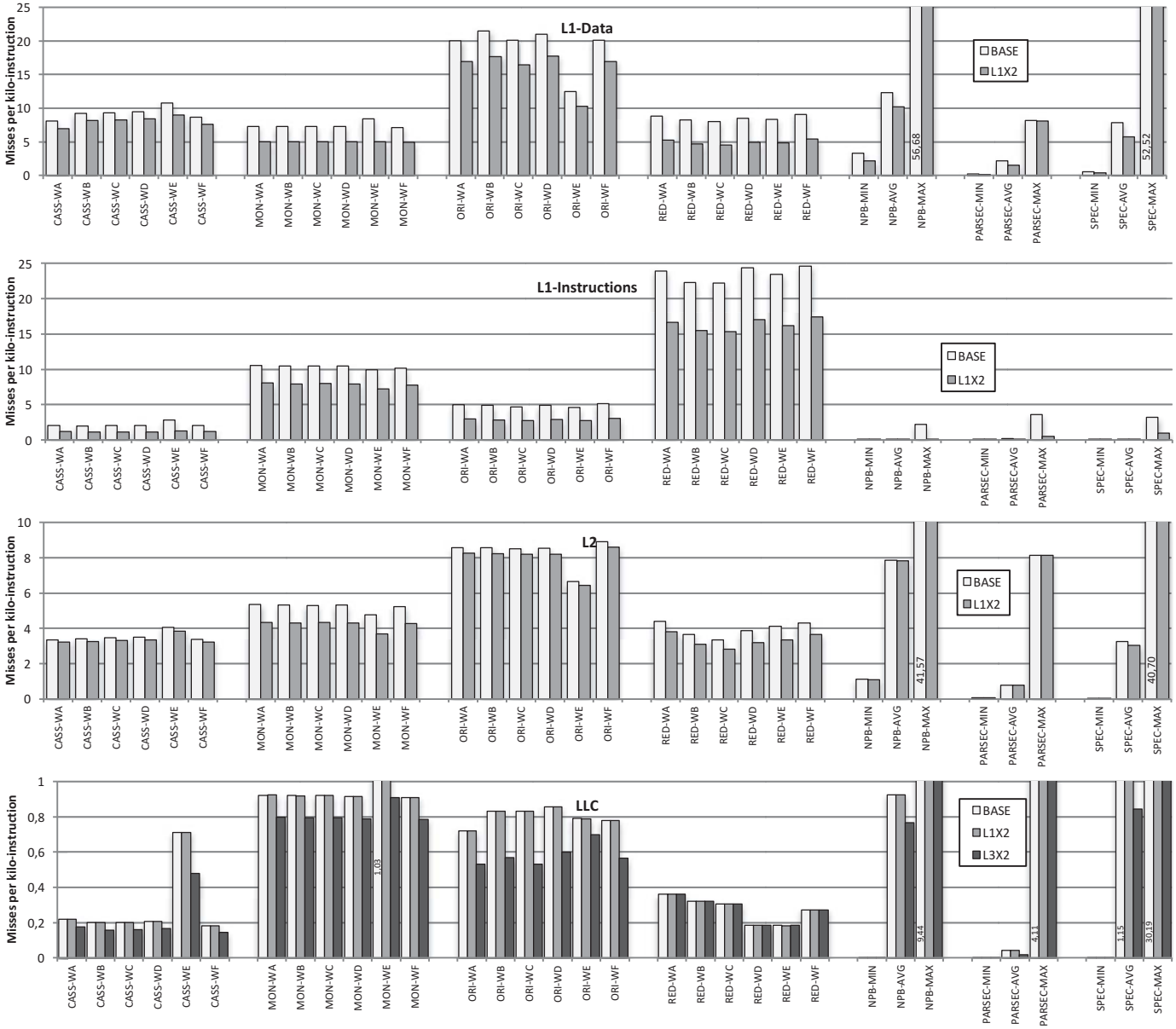

Figure 8. Three-level cache hierarchy performance. Average values of conventional benchmarks are calculated through the geometric mean.

to be the NoSQL data-base engine with least sensitivity to replacement policy, but even in this case, LRU provides better results. In contrast, results show that some conventional applications behave better when RAND policy is applied (SFP-MAX). Analyzing the differences between workloads, all NoSQL applications behave similarly independently of the workload type. Only *MongoDB* seems to be able to extract certain benefit from replacement policy. For the rest of the applications, LRU replacement barely reduces MPKI by 5%.

In contrast to the uniformity observed in L1, the results obtained for LLC show dissimilar results for conventional and NoSQL benchmarks. In the case of conventional benchmarks, in many applications most of the locality has been filtered by private cache levels. The direct consequence is that LRU replacement policy has a much lower impact on this level. This observation has previously been confirmed by many authors, who have demonstrated that replacement policies relying on reuse frequency provide better results than LRU-like ones [35]. Looking at the NoSQL benchmarks' results we observe that in this case, there is still enough temporal locality in LLC to obtain benefits from LRU replacement policy. This result is consistent for three of the four applications evaluated. It is remarkable that both *Cassandra* and *Orientdb* are characterized by much higher MPKI improvements in LLC than those observed in the L1D. These results suggest that it might be necessary to revisit many of the non-LRU replacement policies proposed in the literature. Their performance under a novel set of workloads might not be necessarily like the one observed in conventional workloads (which are the inspiration for the proposals).

C. *Hardware Prefetching*

Through an evaluation process like the one carried out in the previous section, we analyze the spatial locality of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

applications under evaluation. We measure miss-rate values in the presence of simple prefetching policies and compare the results to those obtained without hardware prefetching. The chosen prefetch algorithms are the well-known Tagged Prefetcher [36] and an Arbitrary Stride Prefetcher [37]. Both prefetchers use a degree of 2 blocks (generate prefetches for the next 2 blocks). Figure 10 shows the results obtained for both the first private (L1D) and the last shared (LLC) cache levels. For each experiment, Tagged or Stride policies are applied to the evaluated cache level, while the rest of cache hierarchy does not make use of any prefetching (this means that, in the LLC evaluation there is no prefetcher in L1 caches). Graphs show the results normalized to those obtained by a system with no prefetching at all. To improve figure readability we only provide maximum, minimum and geometric mean values for the conventional benchmarks analyzed.

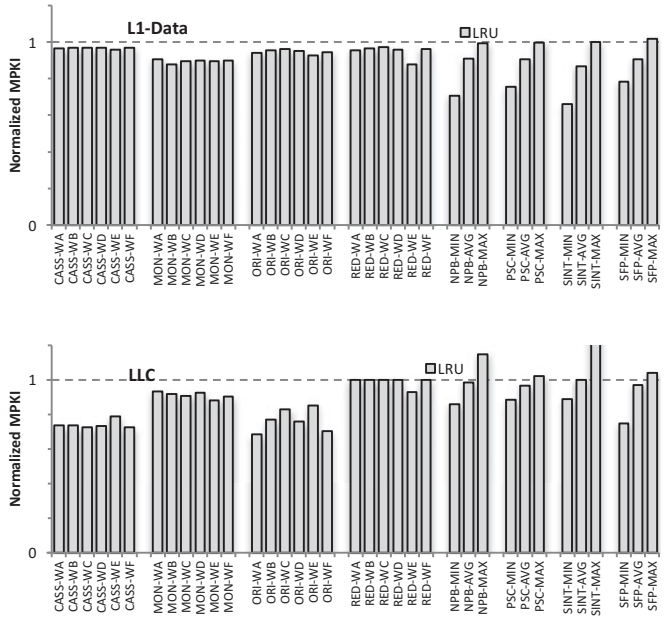

Figure 9. Replacement policy effect on MPKI. LRU results are presented, normalized to RAND values.

The results obtained suggest that spatial-locality is unable to compensate for the pollution introduced in L1D. This is especially true for *Redis* where prefetching, especially the Tagged prefetcher, under-performs the baseline (no prefetching). This is a well-known issue with this aggressive prefetcher, which benefits applications with a high degree of spatial locality, but penalizes applications with little locality, as can be seen in the max and min results of conventional benchmarks. From NoSQL applications, *MongoDB* can outperform MPKI by more than 20% on this level with tagged prefetching and *OrientDB* achieves an improvement of up to 60%, although they seem to be far from the average conventional results. Something similar happens with the prefetcher on LLC, although cache pollution has less impact on this level, and the behavior is generally better. From the results in both figs, it can be concluded that NoSQL applications have a relatively average behavior in relation to prefetching, *MongoDB* being the one that can benefit the most, up to 20% and 60% in L1D and LLC caches respectively, with a consistent behavior among the different workloads.

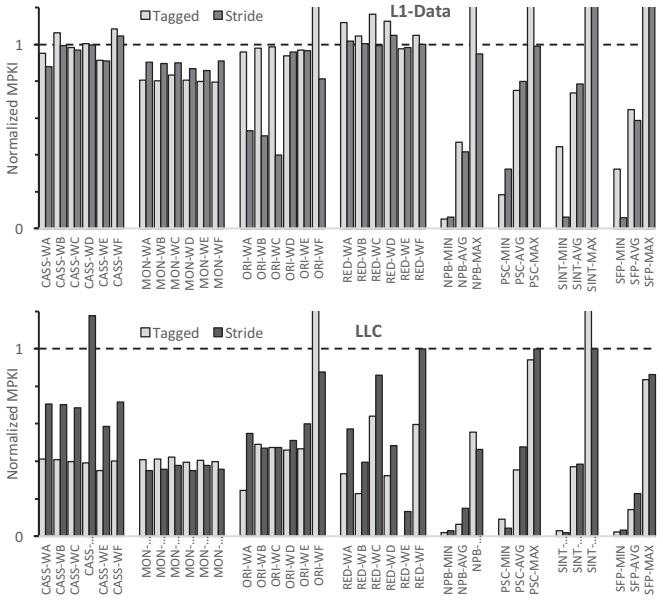

Figure 10. Hardware prefetching effect on MPKI. Results have been normalized to the values obtained in the absence of prefetching.

D. LLC Data & Instruction Sharing

This final experiment focuses on the evaluation of data and instruction sharing degree in the Last Level Cache. Since most coherence protocols propose a scaling mechanism that uses sharing characteristics of common workloads, this study might provide some insights about how they will react with these benchmarks.

To do so, we extend *gem5* to keep track of the different private cache levels "visited" by each block in the cache hierarchy. Sharing statistics are updated at every LLC eviction. Figure 11 shows, for the 4-core configuration employed in previous sections, the fraction of shared and private blocks. The shared blocks are divided according to the number of sharers. We also provide isolated results for instructions and data blocks. As SPEC 2006 applications are all single-threaded, only the results of "conventional" NPB and PARSEC benchmarks are provided.

Comparing NoSQL applications to conventional workloads we can observe some similarities. In both cases, block sharing seems to be much more relevant for instructions. Comparing data and instruction results, we observe that the sharing ratio for instructions at least doubles the values for data. Focusing on instruction results, we observe that *Redis* workloads have a similar behavior to the ones obtained in PARSEC and NPB. In both cases the fraction of non-private instructions is similar, also being dominated by blocks shared by all the available cores in the system (4-SH). *Cassandra*, *MongoDB* and *OrientDB* exhibit different behavior. For these workloads, the fraction of shared blocks is slightly lower and the number of sharers is more uniformly distributed among the three possibilities. Moving on to data-sharing results, we observe different sharing patterns depending on the database evaluated. This variability is also present in PARSEC applications. In contrast, NPB workloads show a minimal

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2017.2787150, IEEE Transactions on Parallel and Distributed Systems

fraction of shared blocks.

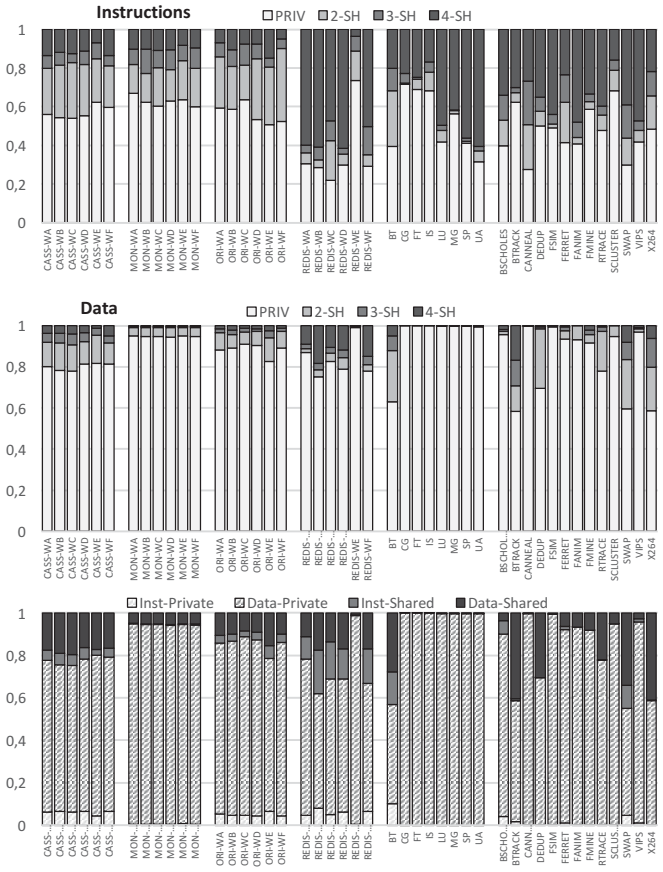

Figure 11. Sharing degree for instructions and data (up and mid graphs). Contribution of instructions and data to private (striped) and shared (solid color) blocks in Last Level Cache.

VI. Related Work

The relevance of Big-data computing systems is visible in the number of studies devoted to the analysis of software applications for such environments [38]. In many cases the absence of well-known benchmarking tools has forced researchers to design their own application suites. Some of these benchmarks, such as *HiBench* [20] or YCSB [15], have been designed to evaluate specific scenarios. *HiBench* was developed to evaluate *Hadoop* [1] performance, while YCSB targets evaluating NoSQL databases. In contrast, other benchmarks such as *CloudSuite* [9], *BigDataBench* [13] and *DCBench* [28] try to cover a wider range of application domains. Both *CloudSuite* and *BigDataBench* have been implemented making use of the full software stack (application and database) and provide workloads targeting domains such as offline analytics, real-time analytics and online services. Alternative multi-domain benchmark suites such as *DCBench* are also available, limiting the software stack to algorithms and synthetic implementations of basic operations.

Making use of all these benchmarking tools, many research works have focused on the microarchitectural characterization and profiling of these scale-out applications [20][9][13][28] [10][11][39][12][40]. Most of these works follow a similar methodology in their evaluation process. Software runs in an actual hardware platform and architectural behavior is analyzed through performance-profiling tools such as perf [30] or VTune [41]. Finally, in some cases the CPI stack is reconstructed following the Top-Down methodology proposed in [42].

Cache Hierarchy results are provided in many of the cited works as part of the characterization process. In [9], the proposed *CloudSuite* benchmark suite is employed to evaluate the inefficiencies of the Intel X5670 processor micro-architecture. L1 and L2 MPKI is evaluated, and a LLC sensitivity analysis is carried out, making use of polluting threads. This work highlights the high instruction-cache miss rate as well as the hardware prefetching inefficiencies detected in our characterization. Trying to overcome methodological limitations (no hardware modifications available), some authors make use of diverse processor models, Xeon E5310 vs. Xeon E5646 in [13] or Xeon E5-2420 vs. Atom C2758 in [12]. In both cases a per-level MPKI analysis is performed, comparing results from both processor models. In [13], the sensitivity of data volume in MPKI is also evaluated, reaching similar conclusions to those in our work, but limited to their specific architectures. All these works make use of extremely heterogeneous workloads and employ a similar methodology based on profiling tools. Focusing on a concrete family of applications and extending characterization through simulation, our work is able to provide much richer conclusions about memory hierarchy effects.

Among all the profiling works, some of them show great similarity to the characterization carried out in this paper. The authors in [11] make use of a similar software stack (YCSB as workload generator) and focus on the same specific kind of applications, that is, modern databases. By making use of profiling tools, performance and scalability of different commercial databases are analyzed. However, again due to the profiling-based methodology, results concerning cache hierarchy are limited to the per-level MPKI analysis. This work claims that the miss-rate observed in L1D cache for NoSQL applications is over 50%, however, we have reproduced these experiments and have not been able to validate this affirmation.

The simulation-based methodology employed in our work represents a key difference compared to all the previous characterization works. This methodology is not new and, in contrast to Big-data benchmark suites, “conventional” suites such as SPEC CPU2006 or PARSEC 3.0 have been characterized previously following a similar methodology to the one employed in this work. In [27] instrumentation-driven simulation is employed to completely characterize the memory behavior of SPEC workloads. In [43], PARSEC applications are also characterized through the simulation tools. However, to the best of our knowledge, this is the first work characterizing the memory hierarchy of NoSQL applications by employing a full system simulation stack.

VII. Conclusions

In this work, we have conducted a simulation-driven characterization of four modern NoSQL databases: Cassandra,

MongoDB, *OrientDB* and *Redis*. We have compared their memory behavior to that observed in benchmark suites designed for micro-architecture evaluation: SPEC CPU2006, PARSEC and NPB. Thanks to the flexibility provided by our methodology, the experiments have allowed us to gain deeper understanding of the cache-hierarchy utilization of NoSQL applications. The results suggest that NoSQL applications present similar behavior to conventional workloads in many cache-hierarchy aspects. Their miss-rate is responsive to cache capacity and performance benefits can be obtained from some hardware mechanisms. In contrast, some other design aspects are unlike conventional benchmarks, and might be relevant enough to require architectural enhancements that take them into account.

Focusing on NoSQL benchmarking, we have found surprisingly uniform behavior in some aspects. The marginal effect of database size on cache performance or the constant fraction of store instructions remains independent of the application or the workload evaluated. The large software stack seems to hide the peculiarities of each application from the underlying micro-architecture. Surprisingly, the dissimilar specifications of each database management software are not reflected in the cache hierarchy. Finally, the characterization work has also highlighted the differences observed between database engines. They present distinctive demands on the memory hierarchy in aspects such as cache capacity, replacement policy or hardware coherence.

The proposed methodology enables a large range of research opportunities. Available workloads could be extended to alternative Big-data benchmark suites, such as *CloudSuite* or *HiBench*. Additionally, the simulated environment enables the evaluation of how microarchitectural proposals will react under these workloads.

REFERENCES

[1] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[2] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1664–1665, Aug. 2009.

[3] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, Feb. 2008.

[4] A. Nayak, A. Poriya, and D. Poojary, "Type of NOSQL Databases and its Comparison with Relational Databases," *Int. J. Appl. Inf. Syst.*, vol. 5, no. 4, pp. 16–19, 2013.

[5] J. Pokorny, "NoSQL databases," in *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services - iiWAS '11*, 2011, p. 278.

[6] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer (Long. Beach. Calif).*, vol. 43, no. 2, pp. 12–14, Feb. 2010.

[7] B. Fitzpatrick, "Distributed caching with memcached," *Linux J.*, vol. 2004, no. 124, p. 5, Aug. 2004.

[8] M. Zaharia, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, and S. Venkataraman, "Apache Spark," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

[9] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *ASPLOS'12*, 2012, vol. 40, no. Asplos, pp. 37–48.

[10] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Performance Characterization of In-Memory Data Analytics on a Modern Cloud Server," in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing (BDCloud)*, 2015, pp. 1–8.

[11] R. Panda, C. Erb, M. LeBeane, J. H. Ryoo, and L. K. John, "Performance Characterization of Modern Databases on Out-of-Order CPUs," in *27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2015, pp. 114–121.

[12] M. Malik, S. Rafatirah, A. Sasan, and H. Homayoun, "System and Architecture Level Characterization of Big Data Applications on Big and Little Core Server Architecture," in *IEEE International Conference on Big Data (Big Data)*, 2015, pp. 85–94.

[13] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "BigDataBench: A big data benchmark suite from internet services," in *Proceedings - International Symposium on High-Performance Computer Architecture*, 2014, pp. 488–499.

[14] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1, 2011.

[15] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, p. 143.

[16] A. Lakshman and P. Malik, "Cassandra," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35, Apr. 2010.

[17] "MongoDB." [Online]. Available: https://www.mongodb.com.

[18] O. T. LTD, "OrientDB," *Available: https://orientdb.com*. [Online]. Available: https://orientdb.com.

[19] S. Sanfilippo, "Redis," *Available: https://redis.io*.

[20] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Lecture Notes in Business Information Processing*, 2011, vol. 74 LNBIP, pp. 209–228.

[21] S. Bast, M. Silva, and N. Wakou, "SPEC Cloud™ IaaS 2016 Benchmark," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering - ICPE '17*, 2017, pp. 423–423.

[22] SPEC Standard Performance Evaluation Corporation, "SPEC 2006," *https://spec.org*. .

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[24] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," *Natl. Aeronaut. Sp. Adm. (NASA), Tech. Rep. NAS-99-011, Moffett Field, USA*, no. October, 1999.

[25] "DB-Engines Ranking." [Online]. Available: http://db-engines.com/en/ranking.

[26] "Apache Cassandra." [Online]. Available: http://cassandra.apache.org.

[27] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," *Web Copy http//www. glue.*

umd. edu/ajaleel/workload, 2010.

[28] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *Proceedings - 2013 IEEE International Symposium on Workload Characterization, IISWC 2013*, 2013, pp. 66–76.

[29] S. Bischoff, A. Sandberg, A. Hansson, D. Sunwoo, A. G. Saidi, M. Horsnell, and B. M. Al-Hashimi, "Flexible and High-Speed System-Level Performance Analysis using Hardware-Accelerated Simulation," vol. 39, no. 2, p. 2012, 2013.

[30] "perf: linux profiling with performance counters." [Online]. Available: https://perf.wiki.kernel.org/.

[31] H. Zhang, B. M. Tudor, G. Chen, and B. C. Ooi, "Efficient in-memory data management," *Proc. VLDB Endow.*, vol. 7, no. 10, pp. 833–836, 2014.

[32] M. Ferdman, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Temporal instruction fetch streaming," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2008, no. 2008 PROCEEDINGS, pp. 1–10.

[33] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive instruction fetch," *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture - MICRO-44 '11*, p. 152, 2011.

[34] C. Kaynak, B. Grot, and B. Falsafi, "Shift," *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchitecture - MICRO-46*, pp. 272–283, 2013.

[35] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *Proc. 37th Annu. Int. Symp. Comput. Archit. - ISCA '10*, p. 60, 2010.

[36] J. D. Gindele, "Buffer block prefetching method," *IBM Tech. Discl. Bull.*, vol. 20, pp. 696–697, 1977.

[37] J.-L. Baer and T.-F. C. T.-F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," *Proc. 1991 ACM/IEEE Conf. Supercomput. (Supercomputing '91)*, pp. 176–186, 1991.

[38] R. Han, Z. Jia, W. Gao, X. Tian, and L. Wang, "Benchmarking Big Data Systems: State-of-the-Art and Future Directions," *arXiv1506.01494 [cs]*, vol. i, pp. 1–9, 2015.

[39] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G. Y. Wei, and D. Brooks, "Profiling a Warehouse-Scale Computer," *IEEE Micro*, vol. 36, no. 3, pp. 54–59, 2016.

[40] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in CloudSuite," in *IISWC 2014 - IEEE International Symposium on Workload Characterization*, 2014, pp. 202–211.

[41] J. Reinders, *VTune Performance Analyzer Essentials*. Intel Press, 2005.

[42] A. Yasin, "A Top-Down method for performance analysis and counters architecture," in *ISPASS 2014 - IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

[43] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," *Proc. Int. Conf. Parallel Archit. Compil. Tech.*, 2008.

Adrian Colaso received his BS and MS in Computing Engineering from the University of Cantabria, Spain, in 2011. He currently has a University of Cantabria grant to work on his PhD in the Computer Architecture group at the same University. His research interests focus on Emerging technologies and applications, as well as their interaction with computer architecture simulation tools.

Pablo Prieto received his BS, MS and PhD degree from the University of Cantabria, Spain, in 2006 and 2014 respectively. He currently works as teaching assistant of Digital System Design for the same University. His research interests are focused on on-chip Cache Hierarchies and Memory Controller design.

Jose Angel Herrero received his BS and MS degree from the University of the Basque Country, Spain, in 2006. He currently works as teaching assistant of System Administration for the University of Cantabria. He also works as System Manager for the Computer Engineering Department of the same University.

Pablo Abad received his BS, MS and PhD degree from the University of Cantabria, Spain, in 2003 and 2010 respectively. He currently works as teaching assistant of Computer Architecture for the Department of Computers and Electronics at the same University. His research interests are focused on the Cache Hierarchy of on-chip multiprocessors, as well as on-chip interconnection network design.

Lucia G. Menezo received her BS and MS from the University of the Basque Country in 2007. In 2014 she received her PhD from the University of Cantabria, where she has worked as a researcher since. Her research interests focus on the memory hierarchy, mainly on cache coherence protocols for chip multiprocessors (CMPs)

Valentin Puente received the BS, MS and PhD degree from the University of Cantabria, Spain, in 1995 and 2000 respectively. He is currently an Associate Professor of Computer Architecture at the Department of Computers and Electronics of the same University. His research interests focus on Memory Hierarchy design and the impact that upcoming technology changes might have on it.

Jose-Angel Gregorio received the BS, MS and PhD in Physics (Electronics) from the University of Cantabria, Spain, in 1978 and 1983 respectively. He is currently a professor of Computer Architecture in the Department of Computers and Electronics in the same University. His main research interests focus on chip multiprocessors (CMPs) with special emphasis on the memory subsystem, interconnection network and coherence protocol of these systems