Peak Performance – Remote Memory Revisited

Hannes Mühleisen
CWI, Amsterdam
The Netherlands
hannes@cwi.nl

Romulo Gonçalves *
IBM Research
USA
goncalves@us.ibm.com

Martin Kersten
CWI, Amsterdam
The Netherlands
mk@cwi.nl

ABSTRACT

Many database systems share a need for large amounts of fast storage. However, economies of scale limit the utility of extending a single machine with an arbitrary amount of memory. The recent broad availability of the zero-copy data transfer protocol RDMA over low-latency and high-throughput network connections such as InfiniBand prompts us to revisit the long-proposed usage of memory provided by remote machines. In this paper, we present a solution to make use of remote memory without manipulation of the operating system, and investigate the impact on database performance.

1. INTRODUCTION

Database systems are using many resources on the computing machinery they run on. Constant disk accesses to guarantee transactional properties and to load persistent data, large amounts of main memory for intermediate result or temporary tables, and moving all data relevant to the query through a CPU all contribute to the usage pattern. Both the amount of data that can be stored in a database ("capacity") as well as the number of queries that can be handled in a time interval ("throughput") are limited by the physical properties of a machine, e.g. the amount of disk space, the size of main memory or the number and speed of available CPUs.

Distributed databases try to solve all these problems at once by distributing both stored data and query execution to multiple machines. Network infrastructure is then used to coordinate these tasks. Similarly, virtual machines (VMs) now have the capability of not only dividing resources of a host computer into many virtual machines, but also to aggregate many physical computers into one large virtual machine. However, data and task placement decisions in these systems always implicate specific weak points, and are thus not fit for every use case [15].

*Most of the research work was done during his PhD at CWI.

DaMoN'13 June 24 2013, New York, NY, USA.

Each of the three resources disk space, main memory and CPU have already enjoyed separate attention with regards to their distribution. For example, the limitations on disk storage space available on a single machine have been overcome by distributed file systems or RAID setups. Similarly, sharing main memory between machines has been proposed frequently for some time now [7]. Considering remote memory is obvious whenever memory is the most scarce resource in a particular system for a particular use case. For example, the recent uptake in main-memory databases with their column-oriented bulk processing paradigm require more memory as the amount of data to be handled grows. For these systems, extending the amount of available memory greatly increases their capabilities.

Using remote memory is only justifiable if performance is competitive with local resources. In particular, remote memory that is slower than the next caching level (typically hard disks) defeats the purpose. However, the recent advent of standardized high-performance network interconnects such as InfiniBand has the potential of profoundly changing this situation. In particular, some of these new interconnects support Remote Direct Memory Access (RDMA) technology, which allows direct access to the main memory of remote machines without involving the CPU or network protocol stack [14]. It is the broad availability of these high-speed interconnects that prompts this revisit of remote memory for databases. In this application area, database queries are often I/O bound, and lower latency and higher throughput have the potential of increasing both query response time as well as overall throughput.

In this paper, we discuss remote memory and its potential benefit to databases. We investigate our main research question whether the broad availability and standardization of RDMA technology can enable usage of remote memory for databases with significant performance benefits. We present a novel solution to make use of remote memory solely based on standard components of the Linux kernel in Section 2. Then, we present a set of experiments from the database perspective to investigate the question of how competitive remote memory is today in Section 3. Furthermore, we discuss the state of the art in using remote memory in Section 4. Finally, we conclude this paper in Section 5.

2. DATABASES AND REMOTE MEMORY

Modern operating systems typically over-commit their physical memory. The memory addresses used by applications refer to *virtual memory*. If the total amount of virtual memory used by all applications outgrows the physical amount of memory in the system, the operating system choses memory pages to be "swapped out" to the hard disk. In most cases, a variant of the least-recently-used (LRU) heuristic is used to decide which pages should be removed. Whenever an application tries to read from an address for which the corresponding page has been swapped out, the operating system will intercept this, read this page from disk again, put it into memory, and the application happily continues. However, there are typically three or four orders of magnitude in difference between disk and memory access latencies. Therefore, if the amount of memory frequently read exceeds the amount of physical memory, "thrashing" will occur, where the system is mostly occupied with swapping pages in and out of memory. This has serious impacts on overall system performance. Therefore, most database systems try to avoid this situation. For example, available memory is frequently used in query optimization cost models to determine the expected execution costs for a query [11].

In relational databases, the amount of memory required to process a single query is also dependent on the processing model used. In the case of an iterator-based approach, only single tuples are processed at a time. High demand for memory can however still occur for example when building hash tables or when temporary tables are being created. Contrary, in the bulk processing model, all intermediate results of operators are materialized in memory. This is done in order to allow high performance in operator execution due to a high probability of CPU cache locality. The advantages of this processing model are particularly apparent when executing OLAP-style queries that require reading a considerable part of the stored data in order to calculate their result. Here, using classical B-Trees to limit the amount of data that has to be loaded from disk into memory is less effective. Furthermore, intermediate results in this class of queries can become very large and even outgrow the size of the explicitly stored data.

Hence, the availability of more memory – or in more general terms, more fast temporary storage – is expected to be beneficial to database performance. However, adding more memory to a single computer quickly loses utility through increasing hardware costs. It is not impossible to order a machine with one Petabyte of main memory, but due to the economies of scale, it is more cost-efficient to purchase a number of smaller machines instead and then enable the computer running the database to make use of the main memory on the other machines. Assuming one is either unable or unwilling to make changes or rewrite applications to distributed operations, one has three main options:

1. Manipulations to the operating system's virtual memory mechanisms, such that pages being swapped out are not sent to local storage, but instead to remote memory. We will discuss previous approaches in Section 4.
2. Since most operating systems allow to specify the location of the area in the file system where pages are swapped out to, one could redirect these to a remote file system using a well-known technology such as NFS and then placing the remote file system in remote memory.
3. If the application actively avoids using the operating system's swap facilities, as it is the case with many databases, one could also redirect their temporary storage areas to the remote memory exported as a file system.

When accessing remote memory through a network interface, it is safe to assume that the memory interface on both sides will not be the limiting factor to access throughput and latency. Moreover, this speed is limited by the throughput and latency of the network interface as well as the device bus speed. This has one important implication: When accessing remote memory through a network interface, the number of remote machines that provide memory is not important. Each remote machine is able to saturate the network and bus bandwidth with data read directly from its memory. Hence, unlike in classical RAID scenarios with slower disks, we can expect full performance regardless of the number of remote machines that are used. Furthermore, comparable to the recent uptake in SSD-based storage solutions due to their negligible performance penalties when performing random access or stride reads, remote memory does not involve moving parts and is thus also expected to excel in this dimension, which is crucial for database operations.

2.1 New Hardware: InfiniBand & RDMA

InfiniBand (IB) is widely used in high-performance computing environments because of higher bandwidth capabilities than Ethernet. Bandwidths of up to 300Gb/s have been specified so far. Furthermore, InfiniBand supports a technology named *Remote Direct Memory Access* (RDMA). RDMA enables direct access to the main memory of a remote host (read and write). The most apparent benefit of using RDMA is a reduction in CPU load thanks to the aforementioned direct data placement (avoid intermediate data copies) and OS bypassing techniques (reduced context switch rate) [1].

A second effect is less obvious: RDMA also significantly reduces the memory bus load as the data is directly DMAed to/from its location in main-memory. Therefore, the data crosses the memory bus only once per transfer. The kernel TCP/IP stack on the other hand requires several such crossings. This may lead to noticeable *contention* on the memory bus under high network I/O. Thus, adding additional CPU cores to the system is *not* a replacement for RDMA.

The RDMA interface is quite different from a conventional TCP socket transfer operations which allows overlapping of communication and computation thereby hiding the network delay. However, RDMA performs best when large chunks of data are transferred. With small transfer units as they occur, e.g., in a tuple-by-tuple transmission, only a small fraction of the available bandwidth can be used. In [5], the authors have shown that, for a single connection, only transfer units of > 4 KB or higher were able to saturate the network link. However, large transfer units come at a price with regards to latency. Hence, a compromise transfer unit of 2 MB is typically used [3].

2.2 Remote Memory for Temporary Files

Instead of manipulating core system services, we have chosen a different approach. Most databases including MonetDB, PostgreSQL and many others allow the database administrator to specify a temporary working directory. In the case of row-oriented databases, this is typically used to materialize temporary tables too large to fit in memory. In the case of MonetDB intermediate results are materialized to allow for bulk processing at every step of the query execution. Here, the temporary directory is also used to hold intermediate results too large for main memory. It is precisely this feature of databases that allows us to take advantage of remote memory without changing core operating system services: Remote access to file systems is a very mature technology, the Network File System (NFS) is a common method to achieve remote file access. We can now piece together a solution for allowing a database to make use of remote memory using RDMA technology as follows:

The nodes providing memory ("Provider") create a *Ramdisk*. A Ramdisk is a virtual file system which is not backed by disk sectors, but instead an area in main memory. To force allocation and allow a later step, we now create a zeroed *Blockfile* that fills the Ramdisk to capacity. Then, the Ramdisk file system is exported using a *NFS Server*. Since NFS supports the RDMA transport protocol, it is this step where we exploit RDMA's advantages as described above. Keep in mind that the number of providers is expected to be larger than one and essentially unlimited, which allows for immense memory-backed file systems.

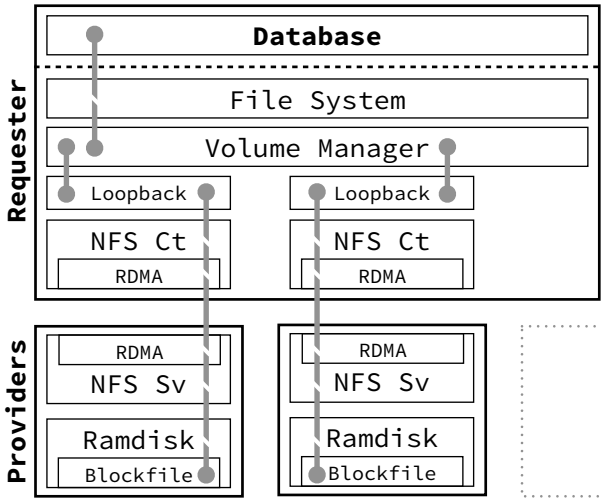

Figure 1: Remote Memory as Local File System

Once this process has been completed on all Providers, the node that likes to use the remote memory ("Requester") now performs the following process: From each Provider, it mounts the NFS share using the *NFS client* with of course the RDMA transport. Then, a *Loopback device* is set up for each of the Blockfiles on the Providers. This allows us to treat the remote Blockfiles that reside in a in-memory file system as a local block device. The set of loopback devices (one for each provider) is now aggregated into a single file system using a *Volume Manager*. In practice, one can either use the Linux Logical Volume Manager (LVM), the Linux Software RAID functionality (`mdadm`), or the multi-volume features of the recent BTRFS file system. In the first two cases, a larger logical volume is created, which can then be formatted with a *File system* (e.g. `ext4`), the latter case does not require this additional layer of indirection and is thus preferred. Finally, the aggregated volume is mounted into the file system, where it can now be used as a temporary directory for a database. This process is visualized in Figure 1. We can see how the database uses the file system to write data to a volume. This volume however is backed by a number of loopback devices, which in turn point to a file on a remote machine, where it resides in a Ramdisk.

While this approach to remote memory might appear cumbersome and slow at first, its huge advantage is not requiring any changes to the operating system, and most recent Linux installations can directly create such a setup "out of the box". Furthermore, an almost arbitrary number of remote machines can be used to provide memory, but through the aggregation in the volume manager, they will appear as a single large file system to the database. Unfortunately, an increasing number of nodes will increase the chance of one of these nodes failing, which would destroy the virtual file system. However, some volume managers support the creation of aggregated volumes with RAID semantics in software, which allows to replace failed nodes and rebuild the virtual volume, even while being in use.

We have created setup scripts for both Providers and Requester, they are available on-line[1]. In the subsequent section, we will investigate on the performance of this approach and see how close it comes to its theoretical performance goal, the raw network interface speed.

3. EXPERIMENTAL RESULTS

We have described our approach in making use of remote memory for fast temporary file storage. Databases, in particular databases that use the bulk processing model and are forced to materialize intermediate results, can draw immense benefits from such a fast file system. Our methodology for our experiments is thus twofold: We first perform baseline throughput and latency tests on the aggregated remote memory. Second, we assess the benefit of remote memory for OLAP database query processing.

All experiments were run on a cluster of 14 machines. Each machine was equipped with 16 GB of main memory, a Intel Core i7 processor clocked at 3.4 Ghz, and a QDR InfiniBand Network card with a theoretical throughput of 40Gb(it)/s. The operating system used was Fedora Linux 16, Kernel version 3.3.4. Of these 14 nodes, 13 were used to provide memory to the 14th. The total amount of aggregated memory on the requester was 182 GB. Keep in mind, that apart from the InfiniBand network these are desktop-class machines.

The technique we have chosen has one crucial advantage: It does not require changes to the Linux kernel. The disadvantage is the rather large number of intermediate components and translations. Even though they all run inside the kernel, they might have such a large performance impact that makes using it impractical. Therefore, we are interested in how

[1] `http://www.cwi.nl/~hannes/rram`

close our method comes towards the theoretical speed limit, which is limited by the speed of the InfiniBand network.

In the second part, we present an experiment where we compare the performance of MonetDB with and without remote memory for temporary storage. As a data set and query workload, we use TPC-H's full set of 22 analytical SQL queries.

3.1 Cluster bandwidth

The cluster is equipped with *QDR* InfiniBand, i.e., a theoretical 40 Gb/s transfer bandwidth. We have used the *qperf* benchmark to determine the realistic uni- and bi- directional bandwidth. For our hardware, we were able to reach a bandwidth of 3.21 GB/s uni-directional and 4.37 GB/s bi-directional. To our surprise, the bi-directional bandwidth is not twice the uni-directional bandwidth. The nodes composing our cluster are commodity desktops which have PCI-16x slots to support graphic cards. Such slots have asymmetric bandwidth contrary to the 8x slots used for servers, which explains the discrepancy. The bi- and uni-directional bandwidth for the two independent servers is an indication that, even with proper slots, the PCI bus rather than the QDR network link is the bottleneck for remote data access. The expected results are 3.8 GB/sec uni-directional bandwidth and 7.6 GB/sec bi-directional bandwidth. Such values are only possible with a PCI 3.0 bus.

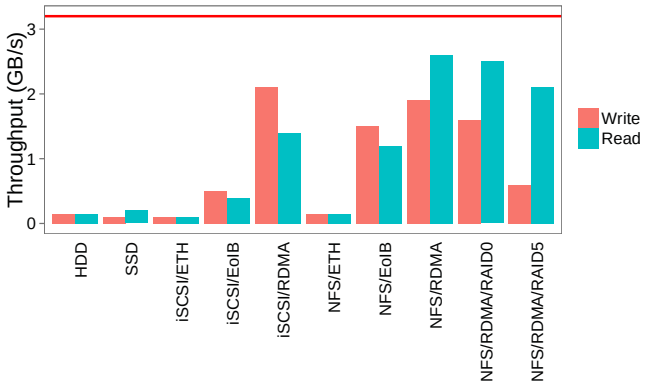

Figure 2: Baseline Performance – Throughput

3.2 Baseline Performance

We have presented a solution for access to remote memory through the file system using NFS and RDMA. In this baseline performance comparison experiment, we have compared our approach to several others with regards to the two main I/O performance indicators throughput and latency: While RDMA promises the highest performance in data transfer compared to IP-bound solutions, we are still interested in how NFS access to remote memory performs when using gigabit Ethernet (ETH) and Ethernet-over-InfiniBand (EoIB). Furthermore, there is another option, the Internet Small Computer System Interface (iSCSI) over RDMA [10]. Contrary to NFS, iSCSI exports block devices and not file systems. Nevertheless, the Linux kernel comes with iSCSI support over Ethernet and RDMA transports. Finally, we compare the performance of our approach when using either RAID0 (striping) or RAID5 (block-level striping with distributed parity) on the volume manager level. As discussed above, we are able to handle failing memory provider nodes with the RAID5 solution, but expect a negative performance impact on writing due to the calculation and writing of the parity blocks. Finally, we introduce control tests where applicable with a local hard disk (HDD), a local Solid State Disk (SSD) and a local Ramdisk. For our RDMA transports, previous work has shown that a message size of 2 MB provides the best compromise between throughput and delay on QDR connections [3]. Hence, this value was used as the block size for our RDMA-backed file systems. For the iSCSI setups, the default message size of 512 bytes was used. For the iSCSI/RDMA setup, the recommended message size of 64 K was configured. The control tests used a Seagate SV35 hard disk and a OCZ Vertex Turbo solid state disk. Both disks were connected using a SATA 2.0 interface.

In order to measure throughput, we have written a 100 GB file to the remote memory volume and measured the time taken. Afterwards, we cleared the file system cache and read the file that was just written again. From this, we can calculate the average throughput rate in Gigabytes per second (GB/s), both for read and write operations. These results can be seen in Figure 2. In addition, this graph shows the measured top throughput of the InfiniBand interface speed as a red horizontal line. We can see how NFS/RDMA clearly outperforms all other solutions in read speed, even with RAID5 redundancy. For writing, NFS over RDMA and iSCSI are similar in performance, and we can also see the huge impact of the parity block writing in the RAID5 setup. Most interesting, the difference between the InfiniBand interface speed and the performance achieved by our approach in reading is rather small, considering the amount of protocol translation involved.

We can observe a similar distribution for the access latency in Figure 3. Here, we have measured the average amount of time to read a block of data from the different file systems. Again, we can observe how RDMA-backed NFS clearly outperforms iSCSI setups and even the local SSD. The difference between the NFS/RDMA setup and the RAID0 array on top of NFS/RDMA are attributed to the software RAID layer, where the `O_DIRECT` flag that disables caching structures had no effect, and requests were thus served out of memory-resident caches.

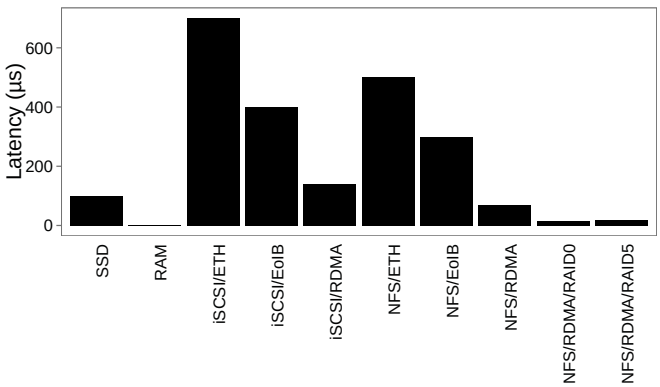

Figure 3: Baseline Performance – Latency

Query	Read (GB)	Write (GB)
q01	13.50	49.44
q18	5.35	28.36
q21	6.74	9.14
q03	5.62	5.96
q13	2.38	6.56

Table 1: Intermediate Traffic - Top 5 Queries

3.3 TPC-H Performance

As discussed in our introduction, memory shortages when storing intermediate results are most serious during analytical query processing within a database. Moreover, in databases using the bulk processing scheme, intermediate results have to be materialized, and if they do not fit into main memory, they are typically moved to the hard disk. In this experiment, we intend to show the relationship between the speed of the temporary storage area in the file system and the query performance. We have chosen the well-known TPC-H benchmark for this purpose. The database under test is MonetDB, which fits well into the described class of systems. MonetDB was extended with a new configuration setting to specify the location for temporary files.

We have set the TPC-H data generator's "scaling factor" to 100, which resulted in a database size of 107 GB. This entire dataset cannot be loaded into the node's 16 GB of main memory. Also, some of these complex queries produce large intermediate results, which do not fit into memory as well. Therefore, they are well suited to show the potential benefits of using remote memory over RDMA for databases. In a preliminary step, we have also measured the amount of temporary data that is written to and read from the temporary storage space for each query. The five queries with the highest amount of temporary I/O are shown in Table 1. From this, we expect the top queries to show the speed differences for the temporary storage area most clearly. The discrepancies between data written and data read in this table stem from the particular implementation of MonetDB, which uses memory-mapped files. Here, the operating system decides when and which changed data is written back to disk.

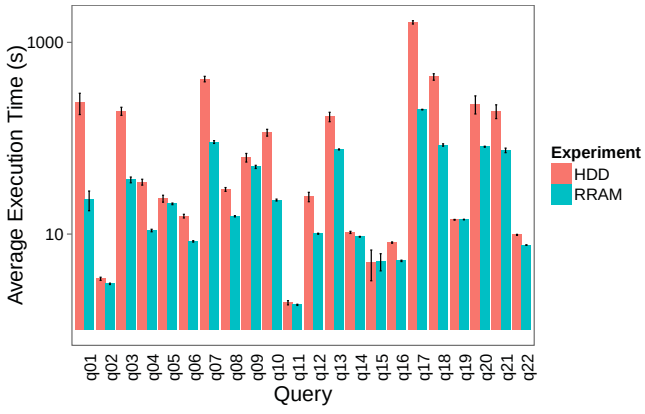

Figure 4: Remote Memory – TPC-H Query Timings

We have run the query workload of 22 queries five times in two environments: "HDD", where the temporary storage area is located on a normal hard disk, and "RRAM", where we have put temporary storage onto a volume backed by the combined memory of our 13 provider nodes using the NFS/RDMA/RAID0 method that showed the best performance in our baseline experiments. The timings reported are cold runs, where the database server has been restarted between queries. For each query run, we have measured the time required to calculate the result as well as the traffic over the InfiniBand interface where applicable.

The timing results from this experiment – averaged over the five runs on a logarithmic scale – are shown in Figure 4. Error bars show the standard error of the average execution time. We can see how the remote memory setup vastly outperforms the baseline HDD performance. Overall, it took about one hour to run all queries with the temporary files being stored on a HDD, but only 15 minutes with the remote memory. If we compare these results again to the temporary storage traffic from Table 1, we can see for example how query 01, which showed the highest temporary storage traffic also shows a large difference in execution times. However, even queries without huge result sets such as 17 showed much improvement. At the same time, the error bars also indicate a lower variance in execution times for the RRAM scenario, which is little surprising since random access is no issue for memory.

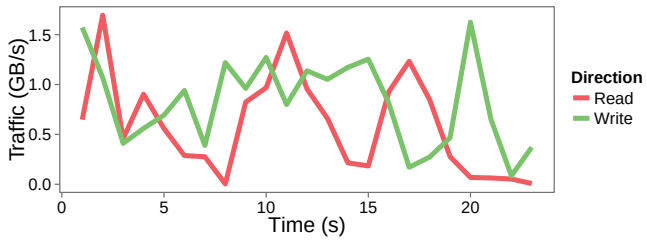

Figure 5: TPC-H q01 Network Traffic

As mentioned, we have also sampled the throughput for the InfiniBand network interface during query executions. Figure 5 shows the interface throughput during execution of Query q01 for both directions. We can see how actual data access by the database reaches a top speed of 1.5 Gigabytes per second, and is sustained for longer periods of time at about one GB/s. While these figures are lower than the bulk throughput rate shown in Figure 2, they are still vastly superior to the throughput achievable on HDDs or SSDs without striping.

4. LOOKING BACK TO LOOK FORWARD

The concept of using to remote memories and high speed networks to improve performance is not recent. Twenty years ago, they were exploited to improve distributed query performance for OLTP systems [7, 4]. In the same line, RAMcloud [13] exploits the fact that remote memory access is becoming more efficient than disk access, i.e., it has lower latency and higher bandwidth.

They observed typical contemporary storage system architectures, like SATA300 and SAS, provide a theoretical maximum IO performance of 3000 Mb/sec (375 MB/sec). Such effective bandwidth is further degraded by additional hardware overhead, e.g., the seek time, operating system software time,

and especially the random access time. Hence, since remote memory does not suffer from seek and rotational delays, access to remote memories is orders of magnitude lower than to the local disk [13].

In the context of distributed file systems, authors in [12] had benchmarked NFS over InfiniBand using RDMA and over Ethernet. The performance results for reading and write were crucial to motivate the study of the RDMA benefits for Hadoop Distributed file system [8] and its further exploitation by HBase [6]. In both cases, due to the architecture of both systems and the use of Java sockets, an extension was required for transparent communication layer. Furthermore, they did not exploit the use of remote memory for storage.

Authors in [2] used a memory based back-end file system for a distributed file system to study, at small scale, the performance and scalability of parallel NFS (pNFS) with PVFS2 as the back-end file system. The idea is to decouple the data and metadata paths and distribute data to multiple storage servers and allow clients to access storage servers in parallel. At the same time, facilitate interoperability, this is, it supports three types of data layouts: blocks, files, and objects.

Standing on the shoulders of this previous work, we plan to explore the use of pNFS and memory based back-end file systems to build a large scale cache for intermediates. Similar to the *sunflower*concept, several DBMSs will sit around the large cache connected through the pNFS-RDMA. Intermediates of one DBMS are re-used to boost the performance of queries settled in the other DBMSs. The concept has been successfully tested for single DBMS and small local cache [9].

5. CONCLUDING REMARKS

Over the course of this paper, we have discussed new uses for remote memory from a database perspective. We have argued for a non-intrusive solution to making remote memory available to a database. We have described our approach, which is based on volume managers, loopback devices, and – most importantly – novel low-latency and high throughput InfiniBand network hardware and the zero-copy RDMA remote memory access protocol. In our experiments, we have seen this solution to come very close to the maximum in network interface speed. Furthermore, in running the standard database benchmark TPC-H on the relational database MonetDB, we were able to see tremendous performance increases.

From our perspective, remote memory through RDMA and mounted into the file system is a great opportunity for speeding up unmodified applications such as databases, which write and read large temporary files to a configurable location in the file system. Our solution can be seen as an "add-on booster" to previous database installations, particularly those that are plagued by lack of memory or the inability to further extend the same. We provide the necessary scripts to set this up in other RDMA-enabled environments with the Linux operating system, and hope to have encouraged the reader to also try our approach.

For now, we manually assign memory providers to a requester. However, due to the immense flexibility of today's volume managers, there are multiple possibilities for future work in this area. For example, one could start with a comparably small volume for temporary file storage by a database, and then periodically monitor this volume with regards to its usage. If the volume usage would cross a certain threshold, one could automatically request a provider node from a predefined pool, connect to it over NFS/RDMA, add the provided Ramdisk to the volume manager, and expand the temporary storage volume on-demand. This solution would also not require any changes to the operating system. Furthermore, nodes could without any issue act as both requester and provider at the same time. A pool of machines could then dynamically share their combined memory, possibly again steered purely by demand. We would expect such a setup to behave particularly well whenever the memory demand on the machines is non-uniform. This approach would also make use of the computing resources on the memory providers.

Another area of future work is swapping to the volume created from remote Ramdisks. We have performed some preliminary experiments on the subject, but have found performance and stability to be non-satisfactory. However, future changes to the Linux swapping mechanism could make this solution viable again. If so, even applications that are not using temporary files for intermediate could directly make use of remote memory.

A valid point of criticism are the current costs of the proposed setup. Infiniband network cards and (more importantly) matching switches are currently very expensive when compared to regular networking hardware. Moreover, the performance figures we have presented could also potentially be reached by combining several solid state drives into a RAID setup. For example, to beat the throughput of around 2.5 GB/s that we have seen in our setup, one would have to combine at least six of the latest solid state drives, which can provide data at a speed of around 500 MB/s (2013-05). Also, a high-price RAID controller capable of operating at these speeds would be required. A different direction of extending the capabilities of a single server would be to extend a single server with more main memory. However, the amount of memory that can be put into a single server is limited by the economies of scale, where only mass-produced main board, processors and memory modules are cost-effective. For example, it is currently not economically viable to purchase servers with much more than 1 TB of main memory, because the ratio between added memory and system price sharply increases after this point.

However, RDMA technology has recently been standardized [14], and there are a number of physical transports already that support RDMA already available, including Ethernet. Hence, there is some indication that the prices for RDMA hardware could drop soon, opening up a new window into fast caches for database operations. Also, high-performance network interconnects also have major advantages for other distributed applications.

Acknowledgments
We would like to thank Arjen de Rijke for rebooting our testing machines whenever we crashed them, Stefan Manegold for help on MonetDB profiling and benchmarking, Holger Pirk for suggesting the "standard system tools" approach and Sjoerd Mullender for his work on MonetDB internals. We also thank the anonymous reviewers for their time and valuable comments. H. Mühleisen is supported by the *COMMIT/* project funded by NWO.

6. REFERENCES

[1] P. Balaji. Sockets vs RDMA interface over 10-gigabit networks: An in-depth analysis of the memory traffic bottleneck. In *In RAIT workshop '04*, 2004.

[2] L. Chai, X. Ouyang, R. Noronha, and D. K. Panda. pNFS/PVFS2 over InfiniBand: early experiences. In *PDSW*, pages 5–11, 2007.

[3] A. Cohen. A performance analysis of 4x infiniband data transfer operations. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 7 pp.–, 2003.

[4] M. D. Flouris and E. P. Markatos. The network ramdisk: Using remote memory on heterogeneous nows. *Cluster Computing*, 2(4):281–293, Oct. 1999.

[5] P. W. Frey, R. Goncalves, M. Kersten, and J. Teubner. A spinning join that does not get dizzy. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 283–292, Washington, DC, USA, 2010. IEEE Computer Society.

[6] J. Huang, X. Ouyang, J. Jose, M. W. ur Rahman, H. Wang, M. Luo, H. Subramoni, C. Murthy, and D. K. Panda. High-performance design of HBase with RDMA over InfiniBand. *Parallel and Distributed Processing Symposium, International*, pages 774–785, 2012.

[7] S. Ioannidis, E. P. Markatos, and J. Sevaslidou. On using network memory to improve the performance of transaction-based systems. In *In International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '98*, 1997.

[8] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda. High performance RDMA-based design of HDFS over InfiniBand. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–35. IEEE Computer Society Press, 2012.

[9] M. Ivanova, M. Kersten, N. Nes, and R. Goncalves. An architecture for recycling intermediates in a column-store. *ACM Transactions on Database Systems*, 35(4), 2010.

[10] M. Ko, M. Chadalapaka, J. Hufferd, U. Elzur, H. Shah, and P. Thaler. Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA). RFC 5046 (Proposed Standard), Oct. 2007.

[11] S. Manegold, P. Boncz, and M. L. Kersten. Generic database cost models for hierarchical memory systems. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 191–202. VLDB Endowment, 2002.

[12] R. Noronha, L. Chai, S. Shepler, and D. K. Panda. Enhancing the performance of NFSv4 with RDMA. In *Proceedings of the Fourth International Workshop on Storage Network Architecture and Parallel I/Os*, SNAPI '07, pages 90–96. IEEE Computer Society, 2007.

[13] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, et al. *The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM*. Stanford University, 2010.

[14] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A Remote Direct Memory Access Protocol Specification. RFC 5040 (Proposed Standard), Oct. 2007.

[15] M. T. Özsu and P. Valduriez. Distributed database systems: Where are we now. *IEEE Computer*, 24:68–78, 1991.