International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:1, 2015

Enhanced Disk-Based Databases Towards Improved Hybrid In-Memory Systems

Samuel Kaspi, Sitalakshmi Venkatraman

***Abstract*—**In-memory database systems are becoming popular due to the availability and affordability of sufficiently large RAM and processors in modern high-end servers with the capacity to manage large in-memory database transactions. While fast and reliable in-memory systems are still being developed to overcome cache misses, CPU/IO bottlenecks and distributed transaction costs, disk-based data stores still serve as the primary persistence. In addition, with the recent growth in multi-tenancy cloud applications and associated security concerns, many organisations consider the trade-offs and continue to require fast and reliable transaction processing of disk-based database systems as an available choice. For these organizations, the only way of increasing throughput is by improving the performance of disk-based concurrency control. This warrants a hybrid database system with the ability to selectively apply an enhanced disk-based data management within the context of in-memory systems that would help improve overall throughput.

The general view is that in-memory systems substantially outperform disk-based systems. We question this assumption and examine how a modified variation of access invariance that we call enhanced memory access, (EMA) can be used to allow very high levels of concurrency in the pre-fetching of data in disk-based systems. We demonstrate how this prefetching in disk-based systems can yield close to in-memory performance, which paves the way for improved hybrid database systems. This paper proposes a novel EMA technique and presents a comparative study between disk-based EMA systems and in-memory systems running on hardware configurations of equivalent power in terms of the number of processors and their speeds. The results of the experiments conducted clearly substantiate that when used in conjunction with all concurrency control mechanisms, EMA can increase the throughput of disk-based systems to levels quite close to those achieved by in-memory system. The promising results of this work show that enhanced disk-based systems facilitate in improving hybrid data management within the broader context of in-memory systems.

***Keywords*—**Concurrency control, disk-based databases, in-memory systems, enhanced memory access (EMA).

I. Introduction

THOUGH main memory sizes have increased dramatically and modern multicore processors provide high-end servers to handle large data sets and computations in memory [1], [2], many considerations such as shared memory access, data partition, persistence and reliability can limit the sole scalability of in-memory databases, especially with drastic growth of big data in businesses [3], [4]. Organisations have to make major decisions to support the recent trend towards in-memory databases and multitenant applications [5]-[7]. Previous works [8], [9] demonstrate that in-memory systems substantially outperform disk-based systems. Thus, it is a general view that for those organizations the best way to improving the performance of their transaction processing systems lies in switching to in-memory systems. However, while in-memory database systems are being developed to support multitenant applications and faster transactions in multicore processing, there are transactional contexts that would continue to use disk-based systems that serve as the primary persistence [10]-[12]. For these organizations, the only way of increasing throughput is by improving the performance of disk-based concurrency control towards achieving better hybrid systems. Literature [13]-[15] shows contention-based scheduler as a way of improving the performance of disk-based systems. This scheduler accepted the generally held view that there is a tradeoff between access invariance and high levels of concurrency. We challenge this assumption and examine how a modified variation of access invariance that we call enhanced memory access, (EMA) can be used to allow very high levels of concurrency in the pre-fetching of data and how this pre-fetching can yield close to in-memory performance in disk-based systems that would lead towards improved hybrid data management systems [16]-[18].

The rest of the paper is organized as follows. Section II describes the proposed enhanced memory access (EMA) principles. Section III presents the implementation of disk-based EMA and Section IV compares the performance of disk-based EMA and in-memory transactions using three popular concurrency control (CC) mechanisms, namely two phase locking (2PL), wait depth limited (WDL) and optimistic CC. The experimental simulation conducted and performance results obtained are analysed in Section V. Finally, Section VI provides the concluding remarks and future work.

II. Proposed Enhanced Memory Access (EMA)

EMA is an extension of the concept of pre-fetching as presented in [14]. Here, all transactions are executed without concurrency control to pre-fetch their data into memory. Once transactions have been pre-fetched, their committable execution can be effected entirely in memory. Fig. 1 illustrates the basic operation of a pre-fetching system.

In Fig. 1, transactions T1 to T10 have successfully pre-fetched and are engaged in committable execution. While these transactions are processing, no memory time is available. However, those transactions that are engaged in disk access to pre-fetch their data can continue in their efforts.

Dr. S. Kaspi is with the Northern Melbourne Institute of TAFE & Melbourne Polytechnic, VIC 3181 Australia (e-mail: samkaspi@melbournepolytechnic.edu.au).

Dr. S. Venkatraman is with the Northern Melbourne Institute of TAFE & Melbourne Polytechnic, VIC 3181 Australia (phone: +613 92691108; fax: +613 92691484; e-mail: sitavenkat@melbournepolytechnic.edu.au).

Should any of transactions T1 to T10 be blocked, then memory time is made available to those transactions requiring such time for their pre-fetching. If any of transactions T1 to T10 completes, then additional memory time is made available to those transactions requiring such time for their pre-fetching – unless there are transactions that have completed their pre-fetching, say transactions T11 to T21, that can take the place of the committing transactions and begin committable execution themselves. Given a sufficient number of available transactions, such a system can yield throughputs near those achieved by in-memory systems.

The major impediment to the success of the process outlined above is that at high concurrencies, access invariance does not hold [19], [20]. That is, at high concurrencies, there is a high probability that the conditions satisfying predicates which determine what data is pre-fetched will change by the time a query is ready to embark on a committing execution [21], [22]. Thus data that has been pre-fetched no longer satisfies the required predicates consequently requiring disk accesses. For this reason, where pre-fetching is used, the number of items pre-fetched is limited.

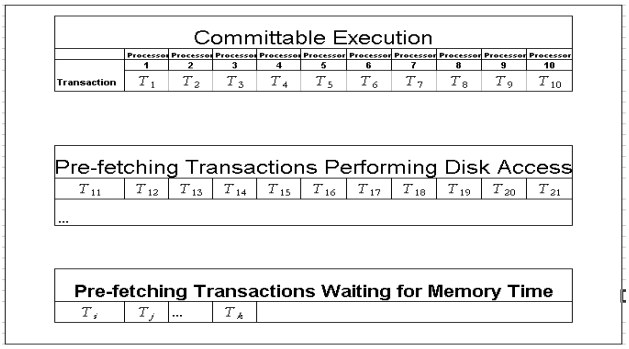

Fig. 1 A sample state for a pre-fetching system

The basis of our proposal for EMA is to allow pre-fetching to the maximum limit physically possible and not to ensure not that conditions satisfying a predicate at pre-fetch time do not change between pre-fetch time and committal execution, but rather, that even where the conditions do change, the data required to satisfy the predicate are also found in memory. While there are several policies and mechanisms that are required for a full implementation of EMA, the basic policy that needs to be implemented is that no data item can be flushed from memory unless the time-stamp of its last access exceeds the timestamp of the oldest transaction in the system. Fig. 2 illustrates the basic principle of EMA.

In Fig. 2, Tk is the newest transaction entering the system - it requires objects Oi and Ok. Object Oi is already in memory having been pre-fetched by transaction T1. Thus, transaction Tk merely updates the time-stamp on object Oi and both pre-fetches and timestamps object Ok. Let us assume that transaction T1 is committing. On committal, it changes the time-stamps and values of the objects that it has used (O1, O8 and Oi). For example, this update changes the timestamp and value of object Oi to reflect that the committal by transaction T1 was the last access to it. Naturally, a committing transaction is the only transaction that can change the value of a data item that is available to all transactions. All other transactions can only modify copies of objects in their own workspace.

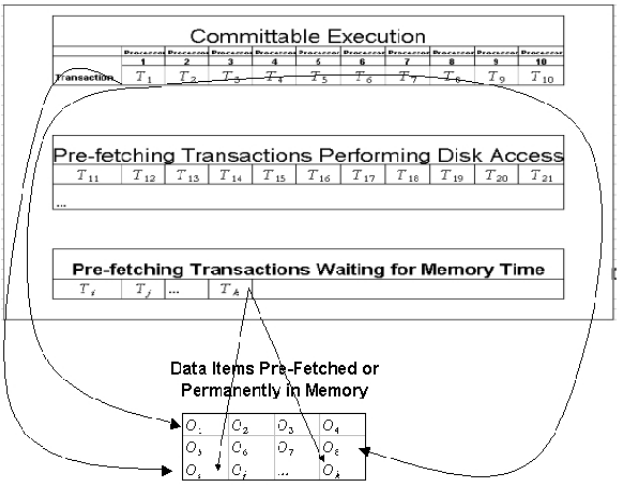

Fig. 2 An illustration of the basic principle behind EMA

Object Oi cannot be flushed from memory until transaction Tk completes since its update time-stamp is newer than transaction Tk's time-stamp. Thus, when transaction Tk begins its committing execution it can access the most recent value of object Oi from memory - (by a committing execution we mean an execution that has the potential to commit rather than merely a pre-fetch execution. We do not mean the act of committal).

The memory requirements of such a system are quite modest relative to the capacity of modern hardware systems. Equation (1) can be used for approximating the maximum possible memory required to support EMA:

$$E = ((F\ n)\ 3\ )\ G \quad (1)$$

In (1), E is memory required, F is the average number of objects required per transaction, n is the permitted concurrency (including pre-fetching) and presuming that the granularity of retrieval into memory is a page, G is the page size (in bytes).

Future database technologies rely on both in-memory as well as disk-based systems that could result in hybrid database technologies using intelligent data management [23]-[25]. In the next section we present an implementation of the proposed EMA that paves the way for such hybrid systems.

III. Implementation of EMA

In order for the EMA to be viable, it requires a mechanism that can perform the following functions economically

1. Determine the youngest transaction in the system at the time a transaction commits,
2. Determine the oldest transaction currently in the system,

3. Record the time of the most recent access for any data object,
4. Flush unwanted data items from memory.

In this section we present one possible implementation of an EMA. Fig. 3 gives an overview of the activity of transactions on entering the system (prior to commencing pre-fetch activity) and how this activity determines the youngest (and in one special case, oldest transaction) in the system at any point in time.

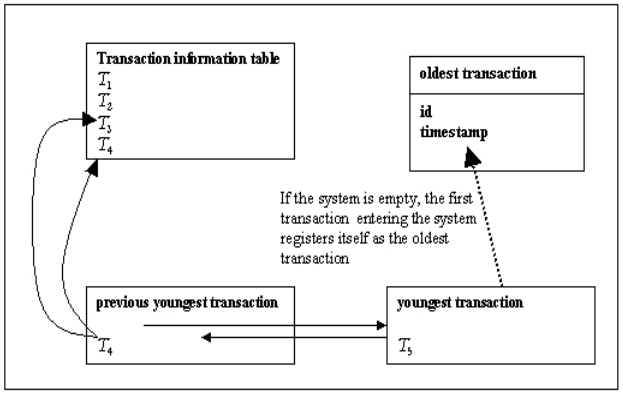

Fig. 3 A typical activity of transaction on entry into an EMA system

As Fig. 3 shows, the system has fixed locations for the oldest, the current youngest and the second youngest transactions in the system. If the system is empty, the first transaction registers itself as the oldest transaction and copies itself to the second youngest transaction's location. From then on, each transaction entering the system follows the same procedure. There are 5 steps involved in this procedure:

1. The transaction copies its details to the youngest transaction location.
2. The transaction accesses the second youngest transaction location and registers its information in the older transaction's next youngest transaction slot.
3. The transaction copies the older transaction's details in its own next oldest transaction slot.
4. The transaction copies the older transaction's information to the memory location indicated by its hash address.
5. The transaction copies itself to the previous youngest location.

Thus, each transaction has access to its next and preceding transaction information and can access this information in a decentralized way – that is, no long queues are formed in accessing this information since the only common resource for which a semaphore is required in this phase is that for the youngest transaction location and there are relatively few instructions required before this resource is freed.

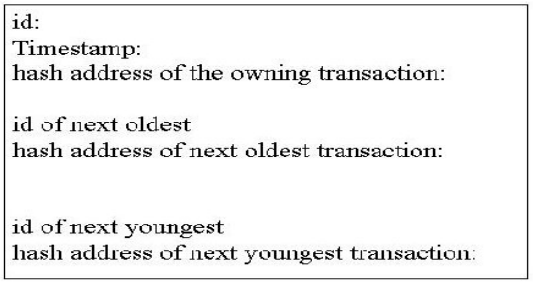

Fig. 4 The composition of transaction markers in the EMA system

The composition of the locations identifying the youngest transaction, second youngest transaction and each transaction's marker in the transaction information table is the same and is shown in Fig. 4. The procedures performed by transactions on transaction markers on completion and exiting from the system are shown in Figs. 5 and 6.

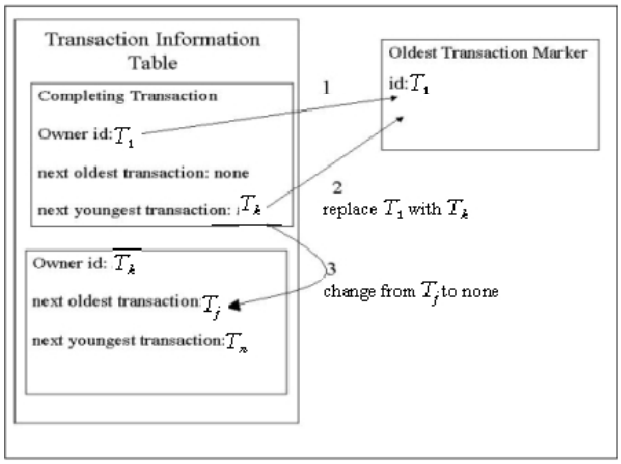

Fig. 5 Exit of the oldest transaction

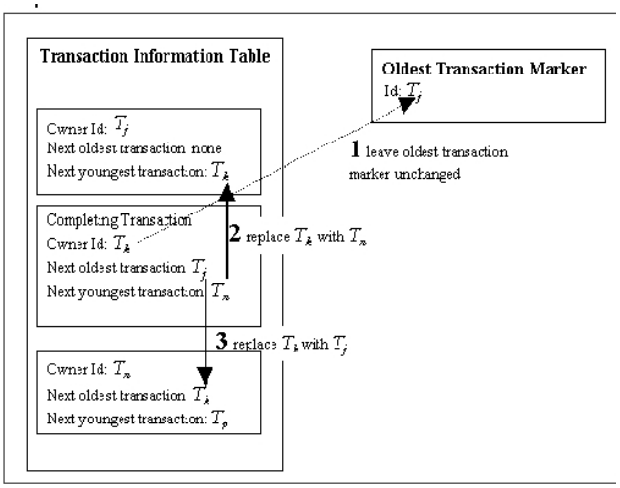

Fig. 6 Exit of a transaction that is not the oldest transaction

In Fig. 6, the exiting transaction, Tk, finds that it is not the oldest transaction and thus does not change the value of the

oldest transaction location. It accesses the transaction marker shown in its older transaction slot, Tj, and replaces the identity shown on Tj's younger transaction slot from, Tk to, Tn. It then accesses its own younger transaction, Tn, and replaces the identity shown on Tn's older transaction slot from Tk to Tj. Tk then deletes itself from the transaction information table. Following the procedures illustrated by Figs. 3-6, it is quite economical to identify both the oldest and youngest transactions at any given moment.

IV. Performance of Disk-Based EMA and In-Memory Systems

In this section, we present three broad series of experimental tests conducted to compare the performance between disk-based EMA systems and in-memory systems running on hardware configurations of equivalent power in terms the number of processors and their speeds. In the first series, we compare three popular concurrency control mechanisms, namely, two phase locking (2PL), wait depth limited (WDL) and optimistic concurrency control (OCC) under EMA and in-memory to each other for equivalent hardware configurations. In the second series of tests, for each hardware configuration, under each concurrency control mechanism, EMA is run under three different costing regimes. In the first two series of tests, where 2PL is used, it is used with thrashing control implemented. Finally, in the third series of tests, we compare the performance of 2PL concurrency control in disk-based EMA and in-memory systems with and without thrashing control.

Under EMA, each transaction is executed twice, once to pre-fetch its data and once to actually commit its actions. With fast processors, the vast majority of a transaction's processing time involves waiting while data is accessed from disk and thus the cost of a double execution is negligible. When optimistic concurrency control is used with EMA we only use optimistic kill concurrency control. We do not use either the die or die-kill optimistic method because the rationale behind their use is that virtual execution allows unsuccessful transactions to pre-fetch their data. Since under EMA data is pre-fetched prior to a transaction's actual execution, further virtual execution merely results in extra wasted work. As well as comparing the performance of in-memory and EMA systems against each other, these three series of tests also compare EMA and in-memory systems to the peak performance of the disk-based optimistic kill method without EMA or access invariance operating at the maximum physical concurrency allowed by the hardware. Optimistic kill operating at the maximum physical concurrency allowed by the hardware is used as a reference as it yields a significantly better performance than any other concurrency control mechanism in the disk-based systems [23]. For each hardware configuration, we present a graph showing the total throughputs achieved. Fig. 7 compares total throughputs in systems containing 20 processors each operating at 1 GIIPS while Fig. 8 shows the experimental results using 10 processors each operating at 2 GIIPS.

From Figs. 7 and 8, we observe that the EMA systems using WDL or optimistic concurrency control reach a hardware-imposed ceiling on throughput. Here, the ceiling is reached after a concurrency of around 10. Because of the low concurrencies involved, the in-memory 2PL system does not reach the ceiling imposed by 2PL's propensity to create wait queues. However, similar to the WDL and optimistic systems, because of the small number of processors available, the 2PL EMA system also approaches the hardware imposed throughput ceiling at a concurrency of around 10.

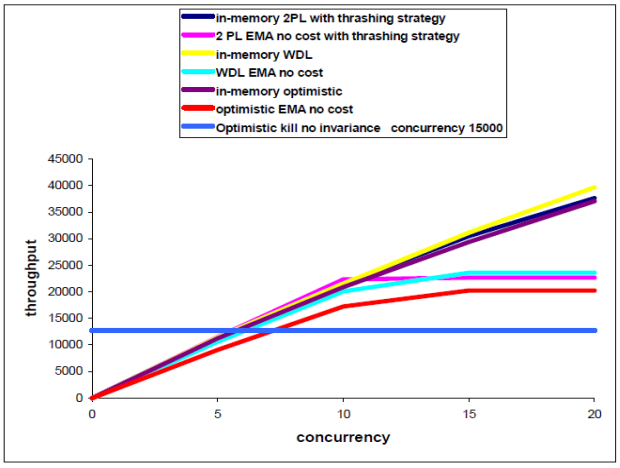

Fig. 7 Performance of disk-based EMA (no cost) & in-memory systems (20 Processors, 1 GIPS)

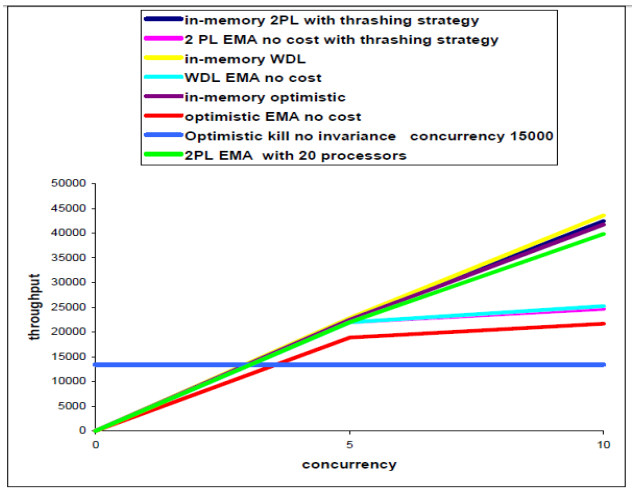

Fig. 8 Performance of disk-based EMA (no cost) & in-memory systems (10 Processors, 2 GIPS)

As an indication of the scalability of EMA, 2PL EMA is also run with 10 extra processors and 15880 extra disks (giving a total of 20 processors at 1 MIPS each and 31760 disks). As shown by Fig. 9, the extra processors and disks bring the performance of 2PL EMA with 20 processors each operating at 1 GIPS, close to that of the in-memory systems with 10 processors each operating at 2 GIPS.

It is noteworthy that for all transactions, WDL using EMA outperforms optimistic concurrency used in conjunction with

EMA. As well, under this hardware configuration, 2PL used in conjunction with EMA also outperforms optimistic concurrency used in conjunction with EMA. As previously explained the main reason between the performance of WDL using EMA (and in this configuration 2PL using EMA) and optimistic concurrency using EMA, is that in optimistic concurrency there is no opportunity for interleaving pre-fetching while a transaction is being executed while under WDL and 2PL, processors with blocked transactions can spend their time usefully by pre-fetching transactions.

The results presented in Figs. 7 and 8 assumed that there is no extra cost involved in administering EMA. In the second series of experimental tests, for each hardware configuration, under each concurrency control mechanism, EMA is run for under three different costing regimes. The first, as in the tests outlined above, assumes a 0 additional cost in implementing EMA. The second costing regime assumes that once its data is pre-fetched, a transaction requires no further disk access. However, it is also assumed that 25% of transactions require forked decisions and that each fork is of equal length. Thus, 25% of transactions require two pre-fetches even though only one of these pre-fetches is used in actual execution. As well, each transaction is penalized 5000 instructions per data item as an overhead cost involved in maintaining EMA. Under the third costing scheme 10% of pre-fetched transactions have to re-access their data from disk once a transaction begins its real execution. Each transaction is penalized 5000 instructions per data item as an overhead cost involved in maintaining EMA. Thus, this series of tests provide a good indication of the robustness of EMA.

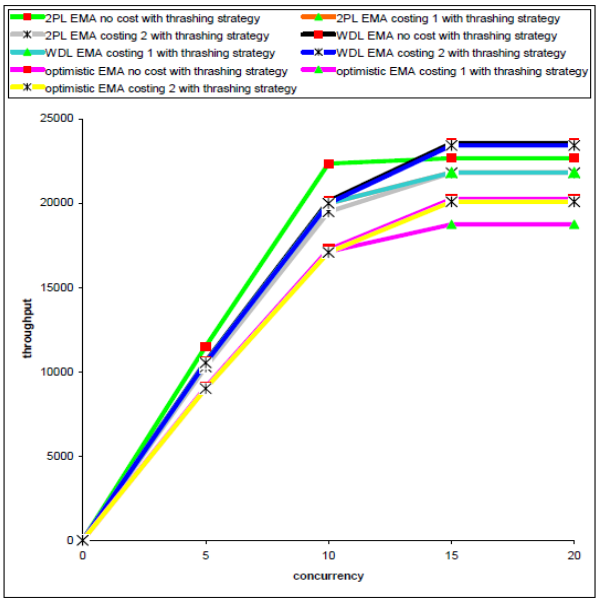

Fig. 9 Performance of disk-based EMA (with costing) & in-memory systems (20 Processors, 1 GIPS)

The results of the second series of tests of EMA are shown in Fig. 9 with systems containing 20 processors each operating at 1 GIPS and Fig. 10 shows the experimental results using 10 processors each operating at 2 GIPS.

Overall, from Figs. 7-10, we observe that since under all tested hardware configurations the addition of significant costs to EMA only affects its performance marginally, one can conclude that this series of tests indicates that the performance of EMA is robust to significant implementation costs.

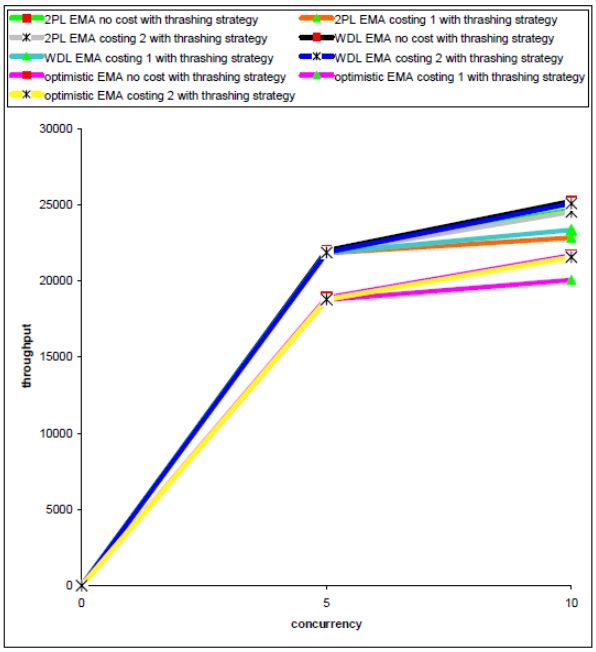

Fig. 10 Performance of disk-based EMA (with costing) & in-memory systems (10 Processors, 2 GIPS)

It is known that 2PL systems are particularly prone to thrashing. This problem increases with contention and concurrency, but even in systems with low concurrencies a very low probability of thrashing at any point in time can still translate to an unacceptable incidence of thrashing given a sufficient number of transactions and processing cycles. In the third series of tests, we compare the performance of in-memory and EMA 2PL systems with and without thrashing control. The thrashing control mechanism used is the immediate re-start of all blocked transactions once the proportion of blocked transactions exceeds 0.378 since this seems to be the most successful method in high-speed systems.

The results of third series of our tests are shown in Fig. 11 with disk-based configurations containing 20 processors operating at 1 GIPS per processor while Fig. 12 presents the results for disk-based configurations containing 10 processors operating at 2 GIPS per processor. Overall, we observe that the advantage of thrashing control diminishes with the reduction of concurrency. Thus in the configurations composed of 20 processors operating at 1GIPS, the advantage of thrashing control only becomes marked in the in-memory system past a concurrency of 15. Similarly, while the advantage of thrashing control for EMA is clear, the performance of EMA is affected more by the ceiling imposed

by the hardware than it is by the implementation of thrashing control. In the configurations composed of 10 processors operating at 2 GIPS, because of the very low concurrencies involved, the effect of thrashing control on throughput is very small and differences in performance are dominated by hardware considerations.

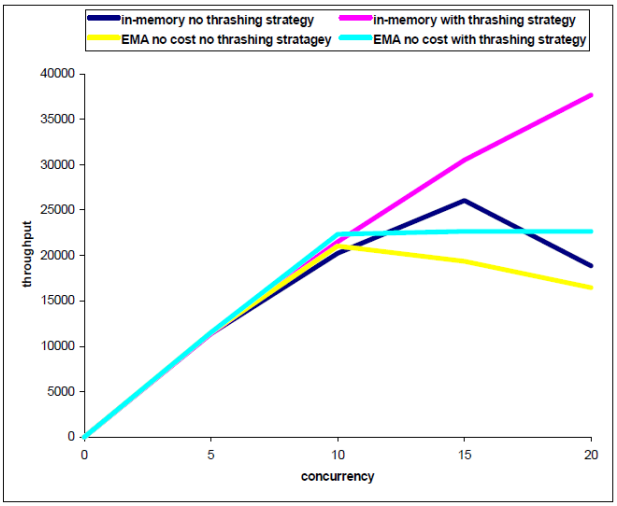

Fig. 11 Performance of disk-based EMA & in-memory systems under 2PL CC (20 Processors, 1 GIPS)

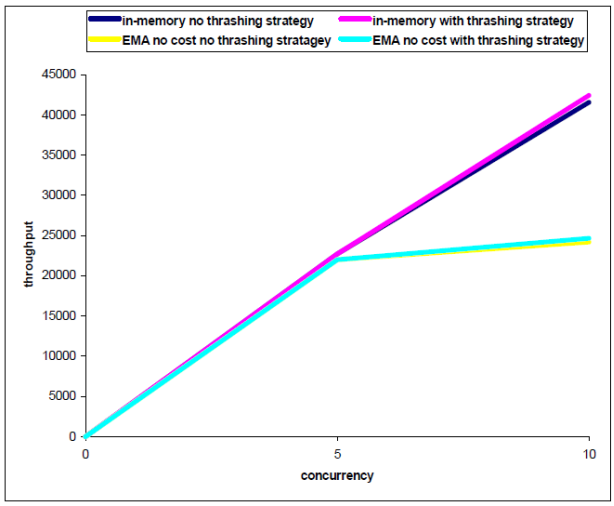

Fig. 12 Performance of disk-based EMA & in-memory systems under 2PL CC (10 Processors, 2 GIPS)

V. Analysis of Results

Our proposed EMA is a variation of the access invariance and pre-fetch schemes outlined in [16], [17]. However, unlike access invariance, EMA does not assume a constant database state or that a transaction will access the same set of objects in all its execution histories. Rather, by controlling the length of time that data resides in memory, it guarantees that having pre-fetched its data, any object that a transaction requires will always be in memory even if the set of data required at actual execution time varies from that established during pre-fetch. This allows a very large number of transactions to be pre-fetched and then executed entirely in memory thus dramatically improving system throughput. EMA improves the performance of disk-based systems (using 2PL, WDL or optimistic concurrency control) to near that achieved by in-memory systems.

The result of the tests presented in this work indicate that when used in conjunction with all concurrency control mechanisms, EMA can increase the throughput of disk-based systems to levels quite close to those achieved by in-memory system using an equivalent concurrency control mechanisms. This performance is far better than can be achieved with any disk-based concurrency control mechanism that does not use EMA. Further, results presented in Figs. 9 and 10 showed that the performance of EMA was very robust to the imposition of additional costs associated with its implementation.

Indeed, the addition of cost penalties far in excess of what one could reasonably expect in commercial applications reduced the performance of EMA by a very small margin. Besides preventing thrashing, the use of thrashing control, improved the performance of 2PL systems. In the systems with the fastest hardware at low concurrency, the performance of the 2PL systems was comparable to that achieved with WDL and optimistic concurrency control. At higher concurrencies, with thrashing control, the performance of 2PL was respectable when compared to WDL. However without thrashing control, the throughput of 2PL actually declined once the peak concurrency threshold was passed.

VI. Conclusions and Future Work

Even though in-memory databases are becoming popular, we believe that disk-based systems would still be in use for several reasons as two different use cases that could be used selectively resulting in hybrid systems. However, to facilitate such hybrid database technologies, in this work, we explored how disk-based database systems could be improved to complement in-memory systems. We introduced a novel disk-based enhanced memory access (EMA) and studied its performance. The main objective of proposing EMA is to allow very high levels of concurrency in the pre-fetching of data thus bringing the performance of disk-based systems close to that achieved by in-memory systems. The basis of our proposed EMA is to ensure that even when conditions satisfying a transaction's predicate change between pre-fetch time and execution time, the data required for satisfying transactions' predicates are still found in memory.

We presented experimental tests that showed that the implementation of EMA allows the performance of disk-based systems to approach the performance achieved by in-memory systems. Further, the tests have also validated that the performance of EMA is very robust to the imposition of additional costs associated with its implementation.

We believe this work opens up future research in the development of new hybrid database system technology that could apply disk-based storage selectively, where certain record types are written to disk, while others are managed

within in-memory databases.

REFERENCES

[1] C. Balkesen, J. Teubner, G. Alonso, and M. T. Özsu. "Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware". In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2013, pp. 362–373.

[2] P. Larson, S. Blanas, C. Diaconu, C. Freedman, J. Patel, and M. Zwilling. "High-performance concurrency control mechanisms for main-memory database". *PVLDB*, 5(4):298–309, 2011.

[3] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. "Saphana database: data management for modern business applications". SIGMOD Rec., vol. 40, no. 4, pp.:45–51, Jan. 2012.

[4] P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, and I. Stoica. "Scalable atomic visibility with RAMP transactions". In ACM SIGMOD Conference, 2014.

[5] D.Jacobs and S. Aulbach. "Ruminations on Multi-Tenant Databases". In Proc. BTW, pp. 514–521, 2007.

[6] V., Ramanathan, S. Venkatraman, and S.R. Asaithambi, "A practical cloud services implementation framework for e-businesses", Book Chapter In Tarnay, K., Xu, L and Imre, S. (Ed.), *Research and Development in E-Business through Service-Oriented Solutions*, IGI Global Publishers, USA, 2013.

[7] B.Mozafari, C. Curino, and S. Madden, "Resource and performance prediction for building a next generation database cloud". *CIDR*, 2013.

[8] S. Kaspi, and S. Venkatraman, "Performance Analysis Of Concurrency Control Mechanisms For OLTP Databases". *International Journal of Information and Education Technology*, 4, 4, pp. 313-318, August 2014.

[9] H. Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD Conference*, 2009.

[10] J. Baker, C. Bond, J. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. L´eon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: "Providing scalable, highly available storage for interactive services". *In Proc. Conf. on Innovative Data Systems Research (CIDR)*, 2011.

[11] I. Petrov, D. Bausch, R. Gottstein, and A. Buchmann, "Data-intensive systems on evolving memory hierarchies," in *Proc. of Workshop Entwicklung energiebewusster Software (EEbS 2012)*, 42. GI Jahrestagung, 2012.

[12] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. "Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration". *Proc. VLDB Endow. (PVLDB)*, vol. 4, no. 8, 2011.

[13] P. Franaszek, J.T Robinson,.and A., Thomasian, "Access Invariance and Its Use in High-Contention Environments", Proceedings of the 6th International Data Engineering Conference, Los Angeles, Feb 1990, pp 47 - 55.

[14] P. Franaszek, J.T. Robinson, and A., Thomasian, "Concurrency Control for High Contention Environments", *ACM TODS*, Vol.17, No.2, June 1992, pp 304 - 345

[15] G. Graefe. "Modern B-Tree Techniques". *Foundations and Trends in Databases*, vol. 3, no. 4, pp. 203–402, 2011.

[16] J. Krueger, C. Tinnefeld, M. Grund, A. Zeier, and H. Plattner. "A case for online mixed workload processing". In *Third International Workshop on Testing Database Systems*, 2010.

[17] J. J. Levandoski, P.-A. Larson, and R. Stoica." Identifying hot and cold data in main-memory databases". In *ICDE*, 2013.

[18] S. Idreos, F. Groffen, N. Nes, S. Manegold, S.Mullender, and M. L. Kersten. "MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 40–45, 2012.

[19] R. Agrawal, M. J. Carey and M. Livny. "Concurrency control performance modeling: Alternatives and implications". ACM Transactions on Database Systems, 12(14): 609–654, 1987.

[20] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[21] P. A. Bernstein and N. Goodman. "Concurrency control in distributed database systems". *ACM Computing Survey*, 13(2):185–221, 1981.

[22] D. Agrawal and S. Sengupta. "Modular synchronization in distributed, multiversion databases: Version control and concurrency control". *IEEE TKDE*, 5, 1993.

[23] S., Kaspi, "Optimizing Transaction throughput in databases via an intelligent scheduler", Proceedings of the 1997 IEEE International Conference on Intelligent *Processing Systems*, Beijing, October, 1337 – 1341, 1997.

[24] C.H.C. Leung, and S. Kaspi, "A flexible paradigm for semantic integration in cooperative heterogeneous databases" *Proceedings of FGCS '94, ICOT*, Tokyo, December 1994.

[25] A., Thomasian, "A performance Comparison of locking methods with limited wait depth", *IEEE Transactions on Knowledge and Data Engineering*, 9(3):421-434, 1997.

Dr. Samuel Kaspi earned his PhD (Computer Science) from Victoria University, a Masters of Computer Science from Monash University and a Bachelor of Economics and Politics from Monash University. He is a member of both the Australian Computer Society (ACS) and Association for Computing Machinery (ACM).

Sam is currently the Information Technology Discipline Leader and Senior Lecturer of IT.at the Department of Higher Education - Business in Northern Melbourne Institute of TAFE (NMIT) and Melbourne Polytechnic, Australia. Prior to joining NMIT, Dr Kaspi taught at Victoria University, consulted privately and was the CIO of OzMiz Pty Ltd.

Sam has been active in both teaching and private enterprise in the areas of software specification, design and development. As chief information officer (CIO) of a small private company he managed the development and submission of five granted and three pending patents. He also managed the submission of a successful Federal Government Comet grant under the Commercialising Emerging Technologies category. He has also had a number of peer reviewed publications including the Institute of Electrical and Electronics Engineers (IEEE).

Dr. Sitalakshmi Venkatraman obtained doctoral degree in Computer Science, from National Institute of Industrial Engineering, India in 1993 and MEd from University of Sheffield, UK in 2001. Prior to this, she had completed MSc in Mathematics in 1985 and MTech in Computer Science in 1987, both from Indian Institute of Technology, Madras, India. This author is Member (M) of IAENG and Senior Member (SM) of IASCIT.

In the past 25 years, Sita's work experience involves both industry and academics - developing turnkey projects for IT industry and teaching a variety of IT courses for tertiary institutions, in India, Singapore, New Zealand, and more recently in Australia since 2007. She currently works as Lecturer (Information Technology) at the Department of Higher Education - Business in Northern Melbourne Institute of TAFE (NMIT) and Melbourne Polytechnic, Australia. She also serves as Member of Register of Experts at Australia's Tertiary Education Quality and Standards Agency (TEQSA).

Sita has published seven book chapters and more than 90 research papers in internationally well-known refereed journals and conferences that include *Information Sciences*, *Journal of Artificial Intelligence in Engineering*, *International Journal of Business Information Systems*, and *Information Management & Computer Security*. She serves as Program Committee Member of several international conferences and Senior Member of professional societies and editorial board of three international journals.