Persistent AutoNUMA

Eduardo Neves Paschoal[1], **Alexandro Baldassin**[1]

[1]Departamento de Estatística, Matemática Aplicada e Computação –
Universidade Estadual Paulista (UNESP) – Rio Claro – SP – Brasil

{eduardo.n.paschoal, alexandro.baldassin}@unesp.br

Abstract. *Persistent Memory (PM) is an emerging memory technology that aims to eliminate the gap between main memory and stable storage. In a conventional Linux system, AutoNUMA is responsible for moving pages between memory regions and processes between hosts to optimize memory latency. For systems with heterogeneous memory types, e.g. DRAM and PM, AutoNUMA provides no support for automatic page balancing. The main reason is that moving pages in a persistent setting requires more attention, as the memory content might become inconsistent in the event of system crashes. This short paper presents our working-in-progress state in adapting AutoNUMA to a persistent environment. We discuss the current status of automatic NUMA balancing in Linux systems and explore some initial ideas to deal with the problem. The main insight behind our approach is to leverage techniques based on transaction processing and journaling.*

Resumo. *A Memória Persistente (PM) é uma tecnologia de memória emergente que visa eliminar a lacuna entre a memória principal e o armazenamento secundário. Em um sistema Linux convencional, o AutoNUMA é responsável por mover páginas entre regiões de memória e processos entre hosts para otimizar o tempo de acesso à memória. Para sistemas com tipos de memória heterogêneos, por exemplo, DRAM e PM, o AutoNUMA não oferece suporte para o balanceamento automático de páginas. A principal razão é que mover páginas em um ambiente persistente exige mais cuidado, pois o conteúdo da memória pode se tornar inconsistente em caso de falhas no sistema. Este breve artigo apresenta o estado atual do nosso trabalho na adaptação do AutoNUMA para um ambiente persistente. Discutimos o status atual do balanceamento automático NUMA em sistemas Linux e exploramos algumas ideias iniciais para lidar com o problema. Nossa abordagem inicial é aproveitar técnicas baseadas em processamento de transações e journaling.*

1. Introduction

New technologies related to main memory have been researched to meet the increasing demand for working with large volumes of data. Areas such as Cloud Computing, Machine Learning, and High-Performance Computing face issues related to the high cost and capacity of current memory technology, DRAM [Lee 2016]. Scalability is a major concern when the limitations of DRAM affect processing capacity, which particularly happens when the dataset exceeds the capacity of RAM [Maruf et al. 2023].

Aiming to reduce costs and achieve ever-higher speeds for data storage and retrieval, new memory architectures have been proposed. Among them, byte-addressable persistent memory (PM) has stood out [Baldassin et al. 2021]. Unlike volatile memory, persistent memory can retain data even in the absence of power. In 2019, Intel launched the first device of this line, known as Intel Optane DC [Tyson 2019]. Subsequently, new technologies such as Compute Express Link (CXL) [Das Sharma et al. 2024] were proposed to encompass these new types of memory. Currently, companies like Numemory are also investing in the development of persistent memory [Mellor 2024].

With the plethora of existing memory technologies co-existing in the same system, it becomes a challenge to dynamically choose at which level of the memory hierarchy a specific data block should reside. When Non-Uniform Memory Access machines started to appear, it was already evident that data placement would become a challenge. Linux started to support Automatic NUMA Balancing back in 2013 to address these limitations. This support is currently known as AutoNUMA [Corbet 2012]. More recently, the support was extended to support also multiple memory tiers [Maruf et al. 2023]. In its current version, however, it does not work with persistent memory since it cannot deal with crash-consistency. For this reason, all non-anonymous memory pages are not handled by AutoNUMA, including pages that refer to persistent memory [Torvalds 2025]. However, several types of applications could benefit from the automatic migration of persistent data in a NUMA environment. One example is large dynamic graph frameworks [Islam and Dai 2023].

This short paper discusses the role of automatic data placement in current Linux systems, shows its limitations in addressing persistent memory, and points to directions on how to extend AutoNUMA to handle it.

2. Background

In typical multiprocessing systems, memory access time is usually not uniform (NUMA) [Lameter 2013]. In this architecture, each set of processor and associated memory forms a NUMA Node. The kernel must allocate memory space for each process using spatial and temporal locality criteria, which typically means using the memory associated with the same NUMA Node where the process is being executed. Until version 3.8, Linux had no efficient support for memory page migration between NUMA nodes, and the community started to research new algorithms to better solve this problem. The module responsible for balancing and distributing memory among NUMA nodes was later introduced to the Linux kernel and called AutoNUMA [Corbet 2012].

Although page migration can highly improve system performance, it can also be expensive. To avoid unnecessary page migration across nodes, AutoNUMA uses some rules to measure whether migration is needed [de Oliveira 2016]. AutoNUMA regularly scans the page table of each process and unmaps its pages by changing the page table entry. When the process tries to access the page, it generates a page fault that is handled by the kernel. While handling this interruption, the OS tracks which NUMA node is accessing the page. If the same node accesses the same memory region twice, the page is migrated [Póvoas 2024].

AutoNUMA's first task was to balance pages across NUMA nodes using the same memory types. Nowadays, AutoNUMA can handle heterogeneous memory divided into tiers: faster but smaller memory in the top tier and slower, larger memory in the bottom tier .

3. Support for Persistency

In the current version of the Linux Kernel, AutoNUMA does not support page migration between persistent memory devices. Since it cannot ensure crash consistency, all non-anonymous memory pages are not managed by AutoNUMA, including those referring to persistent memory, such as pages on Intel Optane DC [Tyson 2019]. Some applications would benefit from this functionality, especially when deploying persistent data structures, such as dynamic graphs. Current solutions to deal with this problem are based on changing the application code to handle NUMA balancing explicitly [Wang et al. 2023]. With the introduction of the CXL standard, this solution obviously does not scale gracefully.

As such, we started to investigate how to extend the current Linux support for NUMA balancing to also take into account persistent memory. One of the major reasons why it does not move persistent data around is due to the fact that it cannot guarantee consistency. Assume for a moment that some data is being moved from one persistent NUMA node to another and a crash occurs (e.g., power failure). On a system restart, the state of the persistent data is likely to be mostly corrupted. To deal with this aspect, techniques based on transaction processing and journaling will be required.

Our initial take on adding automatic NUMA balancing for persistent devices is to use modern Persistent Transactional Systems such as SPHT [Castro et al. 2021]. A transaction is an atomic operation that ensures data consistency because it must either succeed or fail as a single action. When applied to the persistent memory context, we intend to use transactions to ensure atomic page migration. Another similar approach is journaling, which is used by file systems. The latter is used to keep track of changes in files and can recover changes not committed to storage in the event of a system failure. One of the challenges that we envision is dealing with the complexity of the Linux code base. Debugging and testing kernel code is also more complicated than regular code.

As part of the testing infrastructure, we initially intend to use a DRAM+Optane configuration. There will be two Optane memory devices in separate NUMA nodes. Later, we plan to expand the infrastructure to use the new CXL persistent devices. Finally, we intend to collaborate with the Linux community by open-sourcing the code. These are our initial insights for investigating a solution to ensure data integrity, and we hope to elaborate more on this and receive feedback during the event.

4. Conclusion

In this short paper, we briefly discussed the limitations of current memory management systems in Linux, particularly in the context of Persistent Memory (PM). The lack of support in Linux's AutoNUMA utility for consistent migration of persistent data between persistent memory regions has hindered the full potential of these emerging memory technologies. We thus propose to improve AutoNUMA, enabling the migration of persistent data in a consistent manner across different memory devices and processors. By leveraging techniques such as transaction processing and journaling, this approach ensures data consistency, even in the event of system crashes, and facilitates the optimal performance of applications in NUMA environments.

References

Baldassin, A., Barreto, J. a., Castro, D., and Romano, P. (2021). Persistent memory: A survey of programming support and implementations. *ACM Comput. Surv.*, 54(7).

Castro, D., Baldassin, A., Barreto, J., and Romano, P. (2021). SPHT: Scalable persistent hardware transactions. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 155–169. USENIX Association.

Corbet, J. (2012). AutoNUMA: the other approach to numa scheduling. Accessed on April 09, 2025.

Das Sharma, D., Blankenship, R., and Berger, D. (2024). An introduction to the compute express link (CXL) interconnect. *ACM Comput. Surv.*, 56(11).

de Oliveira, M. I. (2016). PTB: An integrated page, thread and bandwidth allocation approach for numa architectures. Master's thesis, Unicamp.

Islam, A. A. R. and Dai, D. (2023). DGAP: Efficient dynamic graph analysis on persistent memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA. Association for Computing Machinery.

Lameter, C. (2013). NUMA (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors. *Queue*, 11(7):40–51.

Lee, S.-H. (2016). Technology scaling challenges and opportunities of memory devices. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 1.1.1–1.1.8.

Maruf, H. A., Wang, H., Dhanotia, A., Weiner, J., Agarwal, N., Bhattacharya, P., Petersen, C., Chowdhury, M., Kanaujia, S., and Chauhan, P. (2023). TPP: Transparent page placement for cxl-enabled tiered-memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 742–755, New York, NY, USA. Association for Computing Machinery.

Mellor, C. (2024). Numemory reinvents optane storage-class memory. Accessed on April 09, 2025.

Póvoas, J. d. L. A. (2024). Optimizing OS Support for Dynamic Page Placement in Heterogeneous Memory Architectures. Master's thesis, Universidade de Lisboa.

Torvalds, L. (2025). torvalds/linux: Linux kernel source tree. Accessed on April 09, 2025.

Tyson, M. (2019). Intel Optane DC persistent memory launched. Accessed on April 09, 2025.

Wang, R., He, S., Zong, W., Li, Y., and Xu, Y. (2023). XPGraph: Xpline-friendly persistent memory graph stores for large-scale evolving graphs. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '22, page 1308–1325. IEEE Press.