Research Statement

Joy Arulraj
Carnegie Mellon University
jarulraj@cs.cmu.edu

Modern data-intensive applications apply statistical analysis algorithms on massive databases to deliver qualitatively better results in many domains, including science, governance, and business. These applications require low latency, always-on, and cost-effective data management. This places increased demands on today's database management systems (DBMSs) whose architectures are tailored for the hardware landscape of the 1980s. The capabilities of nascent hardware technologies invalidate the long-held design assumptions these systems make about memory and storage. This shift argues for research into rearchitecting DBMSs for a new era of heterogeneous, dis-aggregated, and domain-specific hardware architectures.

Given this outlook, the central theme of my research is on *the development of new database management systems that leverage the characteristics of emergent hardware technologies to meet the requirements of data-intensive applications*. In particular, my current research focuses on a new class of non-volatile memory (NVM) technologies that blur the gap between volatile memory and durable storage. NVM supports low latency byte-addressable accesses similar to DRAM, but all writes are persistent like SSDs. There are several aspects of NVM that make existing DBMS architectures inappropriate for them. My research investigates how to rearchitect the DBMS from the ground-up to take advantage of NVM [3]. I redesign the fundamental algorithms and data structures employed in traditional DBMSs to leverage the persistence and performance characteristics of NVM. This enables the DBMS to support low latency transactions, instantaneous recovery from system failures, and cost-effective data management. My work shows that NVM's impact straddles across all the layers of the DBMS, including logging and recovery [8], storage management [6, 11], indexing [9], and query execution [7].

I plan to continue collaborating with partners in both academia and industry to identify shortcomings in existing systems for supporting data-intensive applications. I will take a principled approach to solving these problems by devising analytical models informed by both theory and practice. I will then validate these models by building systems and evaluate their efficacy using scientific experimentation. With the widespread adoption of systems for performing large-scale data science, my research will help move the state-of-the-art forward in this field.

I will next discuss the two primary areas of my current research agenda: non-volatile memory and self-driving DBMSs. These projects are a testament to my overall vision of redesigning DBMSs to take advantage of breakthroughs in other branches of computer science. I conclude with a discussion of the research directions that I intend to pursue in future.

Non-Volatile Memory Database Management Systems

Researchers started exploring the use of NVM in software applications in the late 1980s. However, since they did not have access to hardware, they could only run their prototypes on simulators. As part of our collaboration with the Intel Science and Technology Center for Big Data, I was given early access to an NVM hardware emulator. Since we were the first DBMS researchers to get access to this emulator, I began investigating the behavior of DBMSs on NVM [12]. We can classify DBMSs into two types based on the primary storage location of the database: (1) disk-oriented systems and (2) DRAM-oriented systems. For this study, I ran MySQL and H-Store DBMSs on NVM, which are representative of these two categories. I found that both disk-oriented and DRAM-oriented DBMSs achieve almost the same performance when using NVM because of the computational overhead of their software stacks. They are not ideally suited for NVM since they assume that memory is volatile. This causes them to perform unnecessary work, such as making redundant copies of data on storage to ensure durability. Based on this investigation, I concluded that the best way to resolve these shortcomings is by designing a new system explicitly tailored for NVM.

My research primarily focuses on the development of an NVM DBMS, called **Peloton** [1]. Peloton's architecture illustrates that the impact of NVM spans across all the layers of the DBMS [6–9]. Its design is optimized for data-intensive applications that seek to obtain insights in real-time by analyzing a combination of historical data sets alongside recently collected data. It supports such workloads by concurrently handling transactional updates and analytical queries on the same database. I will next discuss how I adapted each layer in Peloton for NVM.

Logging and Recovery: The ability to quickly propagate data from volatile memory to durable storage determines the performance of DBMSs. Since durable storage devices have high access latencies, particularly for random write

operations, existing DBMSs adopt a logging and recovery algorithm that works around these limitations. They first write out the changes recorded by the applications to a log on durable storage using fast sequential writes. Then, they asynchronously propagate the modifications to the database pages using slow random writes. Most DBMSs use such a *write-ahead logging* (WAL) algorithm. Although WAL improves the throughput of the DBMS, it hurts its availability because the DBMS needs to replay all of the log entries to restore the database after a system failure.

I developed a new lightweight logging and recovery algorithm, called **write-behind logging** (WBL) that leverages NVM's support for fast random writes to improve the availability and performance of the DBMS [8]. With this algorithm, the DBMS first directly propagates the changes out to the database during transaction processing and *later* records *meta-data* about committed transactions in the log. During recovery, the DBMS only read backs the meta-data to determine the uncommitted transactions at the point of system failure. This is sufficient to restore the database back to a consistent state and provides the same recoverability guarantee as WAL.

The results that I published in VLDB show that WBL enables the DBMS to recover nearly instantaneously from system failures, thereby improving Peloton's availability by 100× compared to when it employs WAL. Although WBL also works on other storage devices, Peloton's throughput drops by 8× when it uses WBL instead of WAL on SSDs. In contrast, its throughput *increases* by 30% when running on NVM due to lightweight logging. This is a clear illustration of how NVM fundamentally differs from older storage technologies.

Storage Management: The next area of the DBMS that I investigated was storage management. Existing DBMSs suffer from a high rate of data duplication [12]. When an application inserts a new tuple into the database, the DBMS records the tuple's contents thrice in the table heap, the write-ahead log, and the checkpoint. This write-amplification hurts the performance of the DBMS and increases its operational cost.

To better understand this problem, I implemented three storage engines based on different architectures: (1) in-place updates (e.g., VoltDB), (2) copy-on-write updates (e.g., LMDB), and (3) log-structured updates (e.g., LevelDB). These engines differ in the approaches they adopt for supporting durable database updates. I then designed NVM-aware variants of these engines that make extensive use of a new **non-volatile memory pointer** primitive. Unlike a regular DRAM pointer, an NVM pointer is guaranteed to be consistent after the OS or DBMS restarts. Using this primitive, I built a library of non-volatile data structures that the NVM-aware engines internally use to reduce data duplication [6, 14]. The results that I published in SIGMOD show that these storage managers reduce the response latency by 10×, cut down the DBMS's storage footprint by 60%, and increase its throughput by 5× compared to their traditional counterparts. Such performant and responsive storage engines enable large-scale transaction processing at a fraction of the cost and energy usage.

This improved performance is only achievable when the database is smaller than the amount of non-volatile memory available in the system. However, the databases used by modern data science and machine learning applications exceed the capacity of first-generation NVM products. To support such applications running on large databases, I am designing a new cross-media storage engine that manages data spread across DRAM, NVM, and large-capacity SSDs. I am developing a new class of buffer management policies that, in addition to guiding *what* data should be moved across different devices, also dictate *when* and *to which* device the data should be migrated. The introduction of NVM in the storage hierarchy necessitates these additional design choices. The preliminary results from my evaluation show that by replacing a significant fraction of DRAM with cheaper NVM, we can shrink the operational cost of the DBMS by 8× while still achieving the same performance.

Indexing: After rearchitecting the logging and storage managers of the DBMS for NVM, I shifted my focus to persistent latch-free indexing data structures that support efficient data retrieval. Multi-threaded concurrency is one of the keys to unlocking high performance on modern many-core processors. One particular way to accomplish this in NVM DBMSs is a latch-free tree index. The main drawback of such a data structure is that it is difficult to correctly implement its algorithms since they must handle complex race conditions. One typically implements these data structures using hardware compare-and-swap instructions to modify state. Such instructions only support atomic updates to a single word, but algorithms often require multi-word updates to perform structural modifications. These algorithms work around this limitation by breaking up these multi-word updates into multiple single-word updates. As a result, the algorithms must handle complex race conditions that may occur when other threads observe partial updates.

In collaboration with Microsoft Research, I designed the **BzTree**, a new latch-free index for NVM that overcomes these problems [3]. The BzTree makes use of a *persistent, multi-word, compare-and-swap* (pmwcas) mechanism that atomically and durably installs multi-word updates. By making use of pmwcas, the BzTree does not need additional logic to handle complex race conditions, nor does it require a custom logging and recovery algorithm to guarantee data persistence. The results that I published in VLDB show that the BzTree outperforms Microsoft's prior state-of-the-art

latch-free index (BwTree) by 2× due to its simplified architecture. My cyclomatic complexity analysis of the algorithms in the BzTree show that they are only half as complex as those in the BwTree. This reduces the software development costs associated with testing and extending BzTree.

Although building Peloton from scratch is a challenging task, it allowed me to pursue research directions that would otherwise be difficult had we used an existing system. As the lead graduate student for the Peloton project, I co-advised six master's and four undergraduate student projects based on the system [17]. Peloton started out as a research platform for exploring the implications of NVM for DBMSs. But it is now also being used in pursuing a new research direction on designing a "self-driving" DBMS [15].

Self-Driving Database Management Systems

Tuning modern DBMSs for a particular workload is a laborious and error-prone task due to the long and growing list of knobs that these systems expose. If the DBMS could do automatically tune itself, then it would remove many of the complications and costs involved with its deployment. My research focuses on designing new algorithms that allow Peloton to tune itself. I apply techniques from machine learning to tune the physical design of the database to accelerate query processing. I developed new algorithms for tuning two key components of the database's physical design: (1) storage layout [7] and (2) index configuration [10]. I will next discuss each of them in detail.

Incremental Storage Layout Tuning: To deliver real-time insights, DBMSs need to concurrently handle transactional updates and analytical queries on the same database. But the current trend is to use specialized systems that work well on only one of these workloads and thus requires an organization to maintain separate copies of the database. This adds additional cost to deploying a database application in terms of both storage and administration overhead.

I designed a new algorithm for automatically adapting the storage layout of the database to work well on such hybrid workloads. This layout tuning algorithm first tracks the attributes that queries access. It then uses an online clustering algorithm to determine the frequently co-accessed attributes and derives a workload-optimized layout. Lastly, it incrementally reorganizes the database pages to the new layout. In this manner, the layout tuner amortizes the data reorganization cost across multiple queries.

To support query execution over database pages with different storage layouts, I formulated a new relational operator algebra, called **logical tile algebra**. This algebra enables Peloton's query execution engine to operate on different layouts without sacrificing performance or requiring separate specialized execution engines [7]. The results I presented in SIGMOD show that Peloton delivers 3× higher throughput on dynamic hybrid workloads compared to when it uses static storage layouts. The tuner ensures that the system eventually converges towards the optimal layout for an arbitrary application without requiring any manual tuning.

Predictive Index Configuration Tuning: The next problem I investigated was on automatically determining the set of indexes that the DBMS must build and maintain. Computing such an index configuration is non-trivial because the DBMS must balance the trade-offs between accelerating query execution and reducing index maintenance overhead. Prior research efforts examine the recent query workload in hindsight to determine the set of indexes that they must next build or drop. This retrospective approach towards tuning increases the delay between a workload shift and the associated physical design changes, thereby reducing the utility of the indexes created by the tuner.

I developed a new predictive tuning algorithm that uses reinforcement learning to adapt the database's index configuration. It learns over time to choose which indexes to build and drop in order to maximize the overall utility of the index configuration. I proposed a new **hybrid scan operator** that allows the DBMS to effectively make use of ad-hoc indexes built by the tuner. The results from our evaluation show that Peloton's predictive index tuner improves its throughput on evolving query workloads by 5× compared to a state-of-the-art tuner. Furthermore, the index configuration and storage layout tuners in Peloton work in tandem to incrementally optimize two key components of the database's physical design without requiring any manual tuning. This work is currently under review [10].

Other Research: Beyond my research on NVM and self-driving DBMSs, I have been involved in other projects. I worked on a space-efficient key-value store engine, called SlimDB. SlimDB uses an analytical model to automatically synthesize key-value stores with desired performance and write amplification characteristics [16]. Another project that I am involved with focuses on developing a new query execution engine in Peloton to take advantage of gather-scatter DRAM, a recently proposed hardware technology [13].

Before starting my Ph.D., I examined how to leverage new processor primitives, such as branch-tracing registers and performance counters, to automatically diagnose the cause of software failures using statistical debugging techniques.

My two papers that I presented at ASPLOS showed that these hardware-driven diagnosis frameworks automatically locate the cause of most software failures with less than 3% run-time overhead [4, 5].

Future Research

I plan to continue my research in the field of data management, with a focus on building systems for accelerating data science and lowering the barrier to entry for users. I will accomplish this by leveraging emergent heterogeneous hardware architectures and machine learning algorithms in my work.

Declarative Hardware Management: Future data management systems will operate on hardware architectures with increasing diversity of processors, memories, storage, and networking. These technologies will necessitate a restructuring of commodity DBMSs. As is the case with NVM, their impact will span across all the layers of a DBMS, starting from its log manager all the way up to its query optimizer [3]. Historically, much of the work on rearchitecting DBMSs for nascent hardware technologies have involved adapting a particular system to take advantage of a specific hardware resource. This approach does not scale.

I plan to develop a declarative hardware manager (HM) that translates high-level data processing and storage constructs from different DBMSs into low-level mechanisms for offloading computation and managing storage, respectively. As an example, the storage engine of a DBMS can instruct the HM to organize data in a particular logical layout and order. The HM will then synthesize a physical layout that is tailored to the underlying storage hierarchy. It will make several decisions that are left unspecified in the storage engine's request, such as the distribution of the data across different devices and the customization of the layout for specific technologies.

I foresee that the techniques that I will develop as part of this work can be applied to other problem domains where declarative hardware management could improve a software system's performance and efficiency, such as machine learning systems. For instance, the execution engine in TensorFlow might instruct the HM that it wants to run a matrix multiplication operation. The HM will automatically generate an algorithm for offloading the computation to an appropriate hardware accelerator, such as a GPU. Designing a declarative language and developing a runtime for transparently managing hardware presents a host of exciting research challenges.

Accelerating Data Science Runtimes: Although DBMSs are still used for driving data science applications, analysts also use other languages and libraries. I plan to explore how to apply database technology to accelerate these widely-used data science runtimes. I will map the frequently used data processing operators across these runtimes onto a common intermediate representation in a new general-purpose data science accelerator (DSA). The DSA will optimize the operators' computation and storage within this representation and generate efficient machine code for diverse hardware resources. For example, the storage engine in the R language's default runtime organizes data in a column-oriented layout. However, its linear algebra operators access this data in a row-oriented manner. This mismatch between the storage layout and the access pattern hurts performance due to increased cache misses. Instead, if the runtime offloads the operator to the DSA, the operation will execute on top of a row-oriented layout that the DSA will materialize. As another illustration, after refactoring the execution engine of the Python data analysis library to use the DSA, data science pipelines written in this library will execute faster by using hardware accelerators through the DSA.

Accessible Data Science Interfaces: Even if we accelerate data science pipelines, the benefits of using these systems will reach a wider audience only if their interfaces are easier to use. I have a keen interest in lowering the barrier to entry for users by exploring how to support interactions with such systems using natural languages. Manipulating data science pipelines is a difficult task since this involves programming. While expressive and powerful, structured programming languages are difficult for users without technical training. Even for users with expertise in programming languages, it can be challenging since they need to know which machine learning algorithm to run, gain intuition into the relevant features, and fine-tune algorithmic hyperparameters. Presenting such domain-specific knowledge in the form of helpful hints will make a significant difference in user experience [2]. Designing simpler data science interfaces will be increasingly important as the user base shifts towards non-experts. I will accomplish this by leveraging domain-specific knowledge and machine learning algorithms, such as sequence-to-sequence recurrent neural networks.

The research directions I intend to pursue will require the expertise of researchers in many fields. I look forward to working closely with researchers in systems, machine learning, programming languages, human-computer interaction, and algorithms. I plan to continue collaborating with my industry contacts and researchers outside computer science to identify relevant and interesting research problems, and will use them as a forum for my work to have an impact on real-world applications.

References

[1] Peloton Database Management System. http://pelotondb.org.

[2] SQLCheck: Automated detection of SQL anti-patterns. https://github.com/jarulraj/sqlcheck.

[3] J. Arulraj and A. Pavlo. How to build a non-volatile memory database management system. In *SIGMOD'17: 44th ACM SIGMOD Int'l Conf. on the Management of Data*, 2017.

[4] J. Arulraj, P. Chang, G. Jin, and S. Lu. Production-run software failure diagnosis via hardware performance counters. In *ASPLOS'13: 18th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2013.

[5] J. Arulraj, G. Jin, and S. Lu. Leveraging the short-term memory of hardware to diagnose production-run software failures. In *ASPLOS'14: 19th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2014.

[6] J. Arulraj, A. Pavlo, and S. Dulloor. Let's talk about storage & recovery methods for non-volatile memory database systems. In *SIGMOD'15: 42nd ACM SIGMOD Int'l Conf. on the Management of Data*, 2015.

[7] J. Arulraj, A. Pavlo, and P. Menon. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *SIGMOD'16: 43rd ACM SIGMOD Int'l Conf. on the Management of Data*, 2016.

[8] J. Arulraj, M. Perron, and A. Pavlo. Write-behind logging. *VLDB'17: 43rd Int'l Conf on Very Large Data Bases*, 2017.

[9] J. Arulraj, J. Levandoski, U. F. Minhas, and P.-A. Larson. BzTree: A high-performance latch-free range index for non-volatile memory. In *VLDB'18: 44th Int'l Conf on Very Large Data Bases*, 2018.

[10] J. Arulraj, R. Xian, L. Ma, A. Pavlo, and G. Gibson. Predictive indexing with reinforcement learning. *Under Submission*, 2018.

[11] P. Bailis, C. Fournier, J. Arulraj, and A. Pavlo. Research for practice: distributed consensus and implications of NVM on database management systems. *Communications of the ACM*, 2016.

[12] J. DeBrabant, J. Arulraj, A. Pavlo, M. Stonebraker, S. B. Zdonik, and S. Dulloor. A prolegomenon on OLTP database systems for non-volatile memory. In *ADMS'14: 5th Int'l Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2014.

[13] Q. Li, V. Seshadri, J. Arulraj, A. Pavlo, and T. C. Mowry. Accelerating hybrid workloads on in-memory database systems with gather-scatter dram. *Under Submission*, 2018.

[14] L. Ma, J. Arulraj, S. Zhao, A. Pavlo, S. R. Dulloor, M. J. Giardino, J. Parkhurst, J. L. Gardner, K. Doshi, and S. B. Zdonik. Larger-than-memory data management on modern storage hardware for in-memory OLTP database systems. In *DaMoN'16: 12th Int'l Workshop on Data Management on New Hardware*, 2016.

[15] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *CIDR'17: 8th Int'l Conf. on Innovative Data Management*, 2017.

[16] K. Ren, Q. Zheng, J. Arulraj, and G. Gibson. SlimDB: A space-efficient key-value storage engine for semi-sorted data. In *VLDB'18: 44th Int'l Conf on Very Large Data Bases*, 2018.

[17] Y. Wu, J. Arulraj, J. Lin, R. Xian, and A. Pavlo. An empirical evaluation of in-memory multi-version concurrency control. In *VLDB'17: 43rd Int'l Conf on Very Large Data Bases*, 2017.