TUDOR DAVID RESEARCH STATEMENT

My research interests lie at the crossroads of computer systems and distributed computing. My general aim is to explore new principles, new interfaces, and new system structures that facilitate the design of fast and scalable complex software systems. At a broad level, I am interested in two main directions related to these research areas.

The first is research that investigates how advancements in hardware enable us to design software in new ways. In this area, I have looked at the synchronization primitives that any OS or high-performance system needs, at the kinds of data structures present in modern data stores and concurrent applications, at the communication protocols tomorrow's hardware – which might not have cache coherence – could use, at the changes upcoming memory technologies require from software. I have tried to shine a light on the precise impact hardware features have on these critical pieces of software, and have proposed generic and reusable principles that yield efficient implementations.

The second area I am interested in are large scale, data-intensive, distributed systems. I have investigated protocols enabling high-performance distributed data stores, and, more recently, have started looking at solving challenges related to consensus and fault tolerance in blockchain systems.

While my past work was largely *practical*, much of it relies on, or is presented in relation to, the *theory* of concurrent and distributed computing. I believe this theoretical grounding allows one to design more rigorous and sound systems. In validating claims of an empirical nature, e.g. regarding the performance characteristics of systems, I have taken the approach of an exhaustive experimental validation. I have pursued collaborations with researchers from industry, as evidenced by the projects in collaboration with Microsoft, VMware, and now IBM. I believe that in particular in the area of computer systems, research should not be a purely academic exercise, but should also have *real-world applicability*. In this vein, I have also made the source code related to each of the research projects I have worked on *publicly available*, allowing both the reproduction of results, and, more importantly, reuse of the work. However, I see the main importance of systems research not to lie in any particular piece of software, but rather in the *reusable principles* that show us how complex software systems should be built. Looking into the past few decades of systems research, it is these kinds of principles that have proven to be most valuable and have stood the test of time. In my work, I have also tried to identify such principles, and I see these as the main output of my research.

I believe this focus on practical research, but that is also rooted in theory, on work focused on software, but that is built in close relation to hardware, puts me in an excellent position to tackle some of the most important problems we will need to solve in the following years and perhaps decades.

In my opinion, the main challenge we will have to face in the coming decade is the end of Moore's law. As we will no longer be able to simply rely on a continuously increasing number of transistors, we will have to explore new avenues for speeding up our computing systems. This will of course require us to design hardware in innovative ways, but, just as importantly, will also require us to find new ways of interacting with and taking advantage of this hardware at the software level. I believe the expertise I have accumulated during the past years at this hardware-software interface will enable me to make relevant contributions to this area, and it is here that I would like to focus most of my efforts. I plan to continue studying how new memory technologies can be used, as these have the potential speeding data-intensive applications. In the new world of heterogeneous and specialized hardware, I also want to answer the question of what abstractions should be provided to higher software layers, to render the task of designing fast and portable software systems manageable. In larger scale systems, new networking and storage technologies allow us to structure systems in entirely new ways, with processors and storage not being necessarily grouped in the same unit of failure. While such systems present many advantages, several concerns need to be addressed, such as component failures, consistency of the state, non-uniformity, or appropriate data structures. Moreover, new networking technologies can also provide us with guarantees that enable us to design efficient distributed computing primitives, and I plan to investigate how we can use these to speed up our distributed protocols. Finally, the quest for public blockchain systems that are fast, scalable, robust, energy efficient, and fair is far from over. I plan to extend the work I started in the context of permissioned blockchains to the public blockchain space.

In the rest of this statement, I shall first present the work I have done thus far, after which I will present avenues for future research that interest me.

PAST AND ONGOING WORK

In my past research, most of which was done in the context of my dissertation [2], I have mostly focused on the problem of designing efficient systems for multi-cores, through a number of different perspectives. However, I have also looked at problems of coordination in larger systems. In the following, I discuss the major projects I have worked on and their impact.

Consensus Inside. Scalability and portability issues inherent to shared memory may justify programming the multi-core as a distributed system: processes should assume no shared state and communication should be performed only through explicit message passing. Of course, the communication characteristics inside a multi-core are significantly different from those of a large scale network. In particular, I showed that the computation associated with transmitting a message represents a much more important fraction of the total latency of a message exchange in a multi-core as compared to a LAN environment for instance. Thus, while in large networks limiting the number of round-trips is of paramount importance, in a multi-core limiting the total number of messages that need to be exchanged is the determining factor for performance and scalability. In this context, primitives that are aimed at distributed environments need to be re-thought for the multi-core environment. One such important primitive is agreement. Previous work had only explored blocking agreement protocols (such as Two-Phase Commit) in this scenario. My collaborators and I first showed that a non-blocking algorithm is more appropriate than a blocking one, and proposed using Paxos. This is not necessarily because of failures, but rather in order to be able to tolerate slow processes. We then slightly weakened the guarantees Paxos provides by using only one active acceptor at a time, but in the process significantly reduced the total number of messages exchanged per agreement round. We call the resulting protocol 1Paxos. 1Paxos can make progress as long as at least one of the current leader and the active acceptor are responsive, as opposed to any majority of nodes being responsive. We found this to be a reasonable trade-off, as events requiring extensive fault tolerance that may happen in geo-distributed systems - such as network partitions - simply do not occur in multi-cores. As a result, we were able to significantly improve upon the performance and scalability of alternatives. The resulting paper [7] obtained a Best Paper Award at Middleware 2014.

Study of synchronization. What exactly enables a concurrent object to scale in practice? Much work has been dedicated to the design of scalable synchronization primitives (such as, for example, locks), and higher level concurrent objects. However, the answer to the previous question was still unclear. Was it the nature of the object? Was it the algorithms its synchronization primitives used? Did it have anything to do with the machine the application was executing on, and if yes, to what extent? These are the questions my collaborators and I set to answer. Our approach was to perform an extensive study of synchronization [6]. We attempted completeness along two axes. First, we wanted to consider all layers of synchronization - from the characteristics of the cache coherence protocol, to the atomic instructions provided by the hardware, to synchronization primitives (such as locks and message passing), to higher level concurrent objects (e.g. hash tables), and to concurrent systems (e.g. Memcached). Second, we wanted to explore each of these layers in detail: we considered a large spectrum of architectures with varied characteristics, we studied their cache coherence protocols and all the hardware synchronization instructions they provided, and we implemented a large variety of synchronization primitives, in particular locks. We were thus able to make several interesting observations. In the case of synchronization-intensive software, the hardware - and in particular its non-uniformity - has a significant impact on performance and scalability. In fact, the characteristics of the hardware often prove to be more important than the precise algorithm running on top of it. Concerning the software synchronization primitives, there is no one-size-fits-all solution - practically every synchronization primitive has scenarios where it outperforms all alternatives. Overall, the observations presented in this work can aid concurrent system designers in better anticipating and understanding the behavior of their implementation.

Asynchronized concurrency. Concurrent set implementations are at the core of many important systems. Sometimes, one can use data structure implementations directly provided by a library. However, more often than not, when the goal is a high-performance system, concurrent data structures customized to the need of the particular system need to be designed and implemented. This is a notoriously difficult task, both from a correctness and from a performance standpoint, with no clear picture of how the resulting algorithm should look like. To simplify this task, I identified and proposed Asynchronized concurrency [5]: a set of general patterns that when applied result in highly scalable and portable implementations of concurrent search data structures.

Intuitively, the idea is to design search data structures that resemble their sequential counterparts as much as possible, with stores only to cache lines semantically associated with updated nodes. My collaborators and I showed that it is possible to apply all these patterns to a variety of existing state-of-the-art algorithms, both lock-based and lock-free, and that by applying them we improve the algorithms' performance. In addition, we showcased how new algorithms can be designed starting from these principles. Thus, Asynchronized concurrency offers the programmer both a way to optimize existing concurrent set implementations, and simplifies the task of designing new ones.

Blocking algorithms can be practically wait-free. Another open question in the context of concurrent data structures is the extent to which the theoretical progress guarantee provided by the algorithm matters in practical situations. The stronger the progress guarantee, the more complicated an algorithm is to design. In addition, wait-free algorithms (i.e., the algorithms providing the strongest , and thus theoretically most desirable, progress guarantee) usually suffer in terms performance in the common case. It has previously been shown that a large class of non-blocking algorithms behave like wait-free algorithms in practice. The question I raised was whether blocking implementations can also behave in a practically wait-free manner [4]. One would expect a negative answer given the frequent use of mutual exclusion and the varied conditions under which these algorithms must run. The answer was however *yes* for search data structures, a common class of concurrent objects used to implement the set abstraction. To be able to obtain this answer, I first introduced performance metrics capable of capturing wait-free behavior in practice: the absence of significant delays due to concurrency. I then showed that for virtually all realistic conditions, scalable implementations of blocking concurrent data structures exhibit virtually no delayed requests, and also showed when this conclusion begins to break. I also gave a theoretical intuition for the result, through an analogy with the birthday paradox. Particularly problematic for blocking algorithms were frequent interrupts, in particular context switches. I showed how best-effort Hardware Transactional Memory (HTM) can be leveraged to allow blocking algorithms to maintain practically wait-free behavior in this case. In summary, this work helps programmers better determine when a blocking implementation is sufficient for their needs, thus potentially simplifying the task of implementing a concurrent system to a substantial degree.

Fast and scalable data structures for NVRAM. Non-volatile RAM (NVRAM), an emerging technology, promises latencies and a programming model similar to DRAM, while also ensuring persistence. One of the main challenges when working with NVRAM is ensuring the complete recoverability of a system in case of a sudden restart, while not significantly degrading performance at runtime. This task is additionally complicated by the fact that caches are expected to remain volatile, and we do not control the order in which the data they contain is evicted and written to NVRAM. Existing solutions solve the correctness problem by extensively using costly instructions that write to NVRAM, as well as through logging based on such instructions. Such approaches, we believed, left a lot of room for improvement in terms of performance and scalability. Therefore, in the context of (concurrent) data structures, I proposed *log-free concurrent algorithms*: a set of generic techniques that practically remove the need for logging, and keep write-backs to NVRAM to a minimum [3]. First, I proposed starting from lock-free data structures. A side effect of the theoretical property of lock-freedom is that as long as stores occur in program order, processes never leave the system in an inconsistent state. In the context of NVRAM, this theoretically avoids costly logging operations. Next, I proposed three generic techniques, that show (1) how to correctly implement such durable lock-free algorithms, (2) how to defer doing costly writes to NVRAM in these algorithms, and (3) how to remove logging from the memory management associated with these algorithms. The resulting data structures are in many cases several times faster than previous approaches. Moreover, we applied our techniques on Memcached, an in-memory cache, and we showed that we can maintain its performance, while making its content durable. This allows the system to resume execution immediately after a restart, without having to repopulate the cache. In summary, this work introduces a general methodology for the design of high performance indexes for NVRAM.

Multiversion timestamp locking (MVTL). While ideally, a user interfacing with data can express her computations using individual reads and writes, in the presence of concurrency or failures, it is at times significantly more convenient to group these operations into transactions, which the underlying system must then execute in an atomic manner. Transactions are a basic abstraction for a variety of systems, ranging from key-value stores, to databases, and to software transactional memories. The correct execution of such transactions is ensured through a concurrency control protocol. Such protocols have been extensively studied for decades, with much

recent work being prompted by a resurgence of distributed transactions in large scale data stores. Previous work operates either at the granularity of entire objects, or at least versions of objects. We claim this can unnecessarily limit transaction parallelism, either through excessive waiting, or through excessive aborts. With this in mind, we proposed a new approach for designing concurrency control protocols, called multiversion timestamp locking (MVTL) [1], which operates at the granularity of individual points of logical time. To begin with, I have developed the theoretical foundations of the generic algorithm, and proved that it provides serializability. Next, I have shown that MVTL-based algorithms can commit more transactions than multiversion timestamp ordering algorithms (the closest and most popular alternative to MVTL), can prevent several classes of aborts inherent in previous solutions, and can even emulate many well-known alternatives. In addition, we also developed a key-value store based on MVTL, and showed that in a cloud deployment, it significantly out-performs alternatives operating at the granularity of objects or versions.

Byzantine fault tolerant protocols for blockchains. In the context of my work with IBM Research, I have continued this line of work on larger scale distributed systems, and have been looking at permissioned blockchains (or, as they are sometimes referred to, distributed ledgers). These systems, unlike public blockchains such as Bitcoin or Ethereum, allow only participants with authorized identities, and are generally aimed at business use cases, where several organizations that do not trust each other may want to participate in a blockchain-based consortium to more efficiently perform business transactions among them. Known identities change how such systems are designed in at least one important way: how one can achieve consensus. While public blockchains use solutions such as proof-of-work, which are scalable, but extremely slow and energy hungry, permissioned blockchains can draw on the decades of research in distributed systems and use Byzantine fault tolerant (BFT) protocols. While these protocols are orders of magnitude faster than their counterparts for public blockchains, they have a scalability issue: they are generally considered to be practical only up to a dozen replicas at most. We want to change this belief: we are in the process of designing a BFT protocol that would scale up to 100 nodes. Moreover, we are adapting this protocol to other requirements particular to the blockchain environment. Classical BFT protocols are designed under a model where participants are either correct and willing to help the protocol make progress, or faulty and can take arbitrary actions. In the blockchain environment we can expect to deal with rational participants, who want to maximize their utility. In this context, we are also in the process on integrating an incentive and payment system in the BFT protocol that would incentivize nodes to order client transactions. By achieving all these goals, our aim is not only a step forward for consensus in permissioned blockchains, but a step forward for BFT as a whole in terms of scalability and performance.

FUTURE WORK

My aim is to continue building on the knowledge and expertise I've accumulated during the past years, and apply them to new problems and challenges that are likely to arise in the area of single-machine and distributed systems.

I believe that the foremost challenge in the years to come for computer systems, and perhaps for computer science as a whole, is the end of Moore's law. We can no longer rely on continuously increasing transistor density, and thus, to continue to deliver faster systems, we will need to explore different avenues for speeding up our applications. We will need to rethink how we design and structure not only hardware, which will be more heterogeneous and customizable, and might depart from the von Neummann architecture, but equally importantly, software, as well as the hardware-software interface. On the software side, I believe that the research I have done so far, much of which sits at the interface with hardware, puts me in a good position to tackle such problems, and it is here that I plan to focus the bulk of my attention in my future research.

A second area I am interested in are larger-scale distributed systems. Besides high-performance data stores, which my research has already tackled, I believe that emerging technologies, such as blockchains, that use techniques pioneered by the distributed computing research community and have potential for numerous applications, warrant further attention.

I now present some of my high-level plans in these areas in more detail.

Memory-centric systems. One of the main avenues for increasing system performance is to reduce the latency needed for processors to access data. This has been one of the most active areas of research in hardware design, with several proposals being put forth, many of which change the assumptions we normally make about our computing environment. The question we must now answer is how software can best take advantage of these technologies.

The first such memory technology expected to become a reality is non-volatile RAM (NVRAM), which, ideally, will allow us to access durable data at latencies similar to DRAM. As mentioned in the previous section, I have already started looking at how software should be designed with NVRAM in mind. Nevertheless, we are just beginning to understand the best ways of designing systems around this technology. In an early-stage project, I have begun looking at how to best use both volatile and non-volatile RAM to provide fast access to durable data. Briefly, NVRAM is still somewhat slower than DRAM, and write wearing is a well known issue. The approach I want to propose would result in structures that do the theoretically minimum number of accesses to NVRAM, while maintaining run-time performance, as well as recoverability in case of a restart. I am also interested in investigating other related questions, such as how should the virtual memory system best deal with such memory, or how should a larger (rack) scale system using a pool of NVRAM be structured.

NVRAM is however not the only exciting development in the area of memory systems. Another such technology is near memory computing (or processor-in-memory), in which simpler processing cores that do not communicate through cache coherence are placed closer to memory. A more recent development is in-memory computing, which goes beyond von Neumann architectures, and in which memory cells themselves can be used to perform simple operations. On the longer term, I plan to investigate how software systems should be designed in such a way as to best take advantage of these technologies, and which applications can be adapted to leverage them.

Software for disaggregated hardware. Traditionally, we consider the unit of failure to be a processor with its associated memory and durable storage. However, as a consequence of the introduction of the aforementioned NVRAM, as well as of important improvements in interconnects and network technologies for datacenters, we will be able to directly access remote durable storage extremely efficiently. Thus, we can now place compute and memory components in distinct units of failure, with processors being able to access a large pool of shared and durable fast memory.

While such systems have begun being explored, this area of research is still in its infancy. Challenges include being able to handle machine failures and restarts, ensuring the consistency of the durable state while maintaining performance, designing appropriate data structures, OS-level changes to accommodate these shared abstractions, and dealing with non-uniformity in access latencies. For instance, how should the system deal with a storage component going down? If this should be made transparent to computation, then efficient approaches to replication and fail-over need to be designed. Additionally, what are the best abstractions programmers should use to interact with such storage? Even in my work with NVRAM on a single machine, I have observed a marked difference in interacting with this type of hardware when compared to either DRAM or to block storage. This difference can only be expected to become even more visible when working with NVRAM distributed across several units of failure. Moreover, high-performance systems such as key-value stores or databases will need to use hand-tuned scalable data structures, just as it is the case in other environments today. These data structures need to be optimized for the particular characteristics and non-uniformity of this environment.

Software for heterogeneous platforms. With Moore's law drawing its last breath, another direction hardware innovation is taking is heterogeneity, with architectures including specialized and programmable units. To an extent, this is already visible today, with platforms deployed in datacenters using FPGAs and ASICs. In addition, not all processing units will have access to cache coherence. It is also expected that designing custom hardware to meet one's needs will become simpler.

However, this will result in a much larger diversity of hardware configurations. As such, one can expect software to be closer tied to a particular hardware system. This is a major issue, since it would result not only in software that is burdensome to write, but that is also not portable between different systems. To solve this challenge, we must design appropriate system-level abstractions and communication mechanisms that would facilitate

the design of portable applications for such architectures, yet without sacrificing the performance benefits of specialization. Our work on CosensusInside was a first foray into this area, but I plan to devote more attention to this topic in the future.

Blockchain technologies. As mentioned in the previous section, in the context of my work with IBM Research, I have started looking at blockchain systems. This technology, largely based on concepts which have for a long time been the object of research in the distributed computing community, has been gaining traction as an enabler of trusted, decentralized computing.

Nevertheless, much work is needed in several areas before such technologies can truly be considered practical. Areas of research include security, programming languages, game theory, but also distributed systems. In the latter area, which is where my expertise lies, the quest for consensus that balances fault tolerance, scalability, throughput, latency, and energy efficiency is not yet over. Current solutions for public blockchains either trade one for the others, or are exposed to attacks. Permissioned blockchains offer some solutions, but fundamentally rely on a centralized authority granting access to the system. I plan to continue the work I began at IBM. However, I would like to additionally focus on public blockchains, as solving the issues inherent in these systems would enable a number of important applications. A promising area of research is related to investigating the precise guarantees blockchain applications need from the distributed computing abstractions they use, and optimizing these accordingly.

Distributed systems and algorithms. I would also like to continue looking at distributed protocols and systems for concurrency control and replication. I have studied facets of these problems during my dissertation, but I believe more aspects of the problem remain to be explored.

For instance, recent work has shown how guarantees provided by the network in a datacenter can be leveraged in designing more efficient consensus protocols. I believe more opportunities exist in this area. Many of the inefficiencies of distributed protocols come from the implementation of primitives which are costly, in many instances because few or no assumptions can be made about the underlying network and about the messages we send on it. Networks that provide certain guarantees could be leveraged to greatly improve the efficiency of the software protocols built on top.

Additionally, I believe our work on MVTL can be further extended to develop scalable concurrency control protocols. In our previous work, our main focus was on establishing the theoretical foundations of a genre of protocols that work at the granularity of individual points of logical time. Our next area of focus should now be specific protocols that leverage this paradigm to solve issues present in existing solutions to concurrency control, and integrating the resulting designs into large scale data stores.

Scalable data structures. Scalable data structures have been one of the main areas of research during my dissertation and I plan to continue exploring problems related to this topic. In particular, I have looked at data structures that are commonly used as indexes in data stores. Thus, these structures are key for the overall system behavior. Similarly, the distributed protocols that I have explored are also commonly used in databases and data stores. In this sense, I plan to transfer my insights on scalable data structures and distributed concurrency control protocols to higher-level systems, such as key-value stores and databases. In previous work, I have already looked at optimizing in-memory caches such as Memcached both by making them more scalable, and resilient in the face of restarts. Similarly, one of the classes of systems I want to look at in the future are log-structured merged tress, a class of data stores that can benefit from advancements in both scalable data structure design, and efficient techniques for persistence.

In addition, I plan to look beyond the scenarios explored during my dissertation, and consider new areas where the design of such structures is becoming increasingly important, in particular through collaborations with researchers from the respective fields. For instance, the need for scalable systems and data structures is becoming more stringent in areas such as artificial intelligence. I believe I can apply my expertise in scalable systems in such topics as well.

REFERENCES

[1] Marcos K. Aguilera, Tudor David, Rachid Guerraoui, and Junxiong Wang. Locking Timestamps Versus Locking Objects. PODC 2018.

[2] Tudor David. *Universally Scalable Concurrent Data Structures*. PhD thesis, EPFL, 2017.

[3] Tudor David, Aleksandar Dragojevic, Rachid Guerraoui, and Igor Zablotchi. Log-Free Concurrent Data Structures. Usenix ATC 2018.

[4] Tudor David and Rachid Guerraoui. Concurrent Search Data Structures Can Be Blocking and Practically Wait-Free. SPAA 2016.

[5] Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. Asynchronized Concurrency: The Secret to Scaling Concurrent Search Data Structures. ASPLOS 2015.

[6] Tudor David, Rachid Guerraoui, and Vasileios Trigonakis. Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask. SOSP 2013.

[7] Tudor David, Rachid Guerraoui, and Maysam Yabandeh. Consensus Inside. Middleware 2014.