With the increase in data collection and storage ability of the last decade, computational problems are growing extremely quickly. To enable the kind of bigger computing required for these problems, we distribute tasks to large sets of geographically-dispersed computers. This allows us to apply large amounts of computational power to a single problem, while balancing the load of maintenance and upkeep of the computers. However, distributed computation is in many ways more complex than local and has its own set of unique challenges and foibles.

I work to provide tools that make distributed computing, in all its forms, practical and easy to use. Particularly, I am interested in the properties of distributed data structures, which are fundamental tools in distributed programming. By abstracting basic tasks of storing, sharing, and interacting with data needed by multiple processes, we can remove the complications of coordinating concurrent operations from a programmer’s burden. I wish to explore the possibilities of distributed data structures, from implementation to specification, speed and computational power. To do this, I work on developing efficient algorithms for data structures, specifying and designing properties of data structures that are either useful to a user or allow more efficient implementation. Conversely, I explore lower bounds and impossibility results for distributed data. By knowing the limits of what we can do, we know where to look for improvements.

Relaxed Data Types

Much of my work so far has focused on relaxed data types. In essence, relaxing a data type adds a precise, limited amount of non-determinism, which allows different processes to perform some concurrent operations with less synchronization than is required for fully deterministic data types. Since communication delays are often vastly larger than those of local computation, this translates to significant performance improvements. For example, one type of relaxed FIFO queue allows each \texttt{Dequeue} to return one of the $k$ oldest elements currently in the queue, instead of the exact oldest. This can allow up to $k$ \texttt{Dequeues} to be performed without any synchronization between processes.

We have designed distributed algorithms to implement two versions of a relaxed queue. We proved lower bounds on the amortized time of any implementation of an unrelaxed FIFO queue that show our algorithms have better performance, in an amortized sense, than any possible algorithm for an unrelaxed queue. We also proved lower bounds on algorithms implementing these relaxed queues. These showed, first, that our algorithms were asymptotically optimal and, second, that increasing the relaxation parameter gave continual improvements to performance. This work confirms that we should be looking for applications which can use relaxed data types, since they can achieve better performance than the corresponding unrelaxed types.

Another historically common approach to improving the performance of distributed data structures is to weaken the required consistency condition. Consistency conditions specify how concurrent behavior relates to sequential data type specifications. By weakening the constraints on concurrent behavior, more distributed executions are considered acceptable, and more efficient algorithms become possible. We have shown that data type relaxations are a subset of consistency conditions. That is, relaxations and some weakened consistency conditions are different ways to express the same set of allowable behaviors. We show correspondences between specific conditions and use existing tools from the literature to compare the strength of some relaxations by comparing their corresponding consistency conditions. This ability to use whichever model is more convenient is one direct, practical benefit of the correspondence.
Intuitively, relaxation gives up some strength to allow greater efficiency. We used the notion of consensus numbers to characterize the amount of computational strength lost through relaxation. The consensus number of a data type, which is defined as the largest number of processes which can use that type to solve the consensus agreement problem, is a classic measure of the computational strength of data types in asynchronous, failure-prone systems. We considered three relaxed versions of augmented queues, which have a *Peek* operation, as well as *Enqueue* and *Dequeue*. Each operation can be relaxed by an integer parameter, so we have an infinite 3-dimensional space of relaxed queues for each relaxation. By directly proving the consensus numbers of a few specific relaxations and some lemmas relating the consensus numbers of different choices of relaxation parameters, we found exact consensus numbers for every point in these relaxation spaces. Our results show that even a slight change in relaxation can significantly reduce the computational power of the type. This shows that it behooves a developer using a relaxed data type to choose a relaxation type and parameters very carefully to ensure the data type maintains the required computational strength.

**Sensitivity**

One of the primary concerns when working with distributed data types, relaxed or unrelaxed, is their computational strength, most often represented by the type’s consensus number. We introduced the notion of *sensitivity*, which captures which part or parts of the prior history of operations on the data structure a process can learn about in a single operation. We then proved consensus numbers for data types with operations in one of several classes defined by sensitivity. This allows a user to find a data type’s consensus number simply by determining its sensitivity class. This can be much easier than directly finding a consensus number by giving an algorithm to solve consensus among a certain number of processes and an impossibility proof showing that no such algorithm exists for any larger number of processes. Our new method may thus help a developer who needs to quickly determine the computational power of a given data type.

**Future Work**

I would like to continue to work with relaxed data types. So far, researchers have usually paired relaxed data types with linearizability, to ease understanding. By pairing data type relaxations with weaker consistency conditions, we may be able to extend the partial equivalence between relaxations and consistency conditions. We may also be able to use tools from both worlds to more easily design efficient implementations or prove better lower bounds.

I also intend to continue to try to improve implementations of relaxed data types. One direction that needs to be explored is more practical model assumptions, such as asynchrony and various types of failures. Some researchers have worked in this direction, but often without theoretical analysis of the efficiency of their implementations. I am interested in finding provably-efficient implementations and matching lower bounds to show optimality in realistic system models.

This also leads to a great opportunity to work with students and researchers with backgrounds in a variety of disciplines. With efficient real-world implementations of relaxed data types, we can use their greater efficiency to solve problems in a wide variety of domains. Domain-specific knowledge will also provide insight on what relaxations on which types are most useful to explore. This interaction is the ultimate goal of research on relaxed data types, and I am excited to work with other researchers to find and solve new, interesting problems.