To effectively support parallel computing involving shared data, the issue of ease of using shared data in a data-intensive parallel computing environment must be addressed. An underlying system facility should be available which provides flexible and intelligent services on the coordination of accesses to the shared data, but does not impose constraints on the conceptual model of the data objects or the functional operators that might be used to manipulate these data, or the programming language in which these operators may be expressed.

We are investigating an approach to transparently supporting data sharing in a loosely coupled parallel computing environment, where a moderate to a large number of individual computing elements are connected via a high-bandwidth network without necessarily physically sharing memory. Although there is no physically shared memory, our approach allows programmers to view the shared data as residing in the programs' virtual memory address space. Multiple threads of execution of these programs share portions of the virtual space, yet the execution may take place on different computing elements in the network, and can directly reference and manipulate the shared data in the shared virtual space concurrently. This distributed shared virtual memory is supervised by an underlying system facility which is transparent to the programs sharing the data and is responsible for data migration and data integrity. We believe that this shared memory paradigm is a viable alternative to the message passing paradigm, and offers convenience for programming data-intensive parallel computation applications such as collaborating computing tasks.

We are building a system called VOYAGER which serves as the underlying system facility that supervises the distributed shared virtual memory. VOYAGER supports the notion of transactions. Programs that access the shared virtual memory do so by bracketing the accesses into transactions, and VOYAGER ensures that transactions see only consistent shared data states, that committed updates to the shared data are made to persist, and failed transactions do not contaminate the share data. VOYAGER also allows some designated portions of the virtual memory to be shared with a mechanism weaker than that using transactions, allowing the possibility of trading services automatically provided in transactions for higher performance. VOYAGER takes care of distributed concurrency control, persistence of shared data, rollback of failed transactions, and migration and caching of shared data in the network. It performs these tasks transparently by supervising references made to the shared virtual memory during the execution of transactions or threads, analogous to the transparent manner in which a conventional virtual memory manager provides its paging services. Therefore the services provided by VOYAGER are equally available to programs written in different languages and employing different data models or object models, as long as the virtual address space where the shared data resides is properly defined.

VOYAGER allows shared-data parallel applications to take advantage of parallel and distributed processing with relative ease. The application program merely maps the shared data onto its virtual address space and replicates itself on distributed machines and spawns appropriate execution threads; the threads would automatically be given coordinated access to the shared data distributed in the network. Multiple computation threads migrate and populate the processors of a number of computing elements, making use of the multiple processors to achieve a high degree of parallelism. The low-level resource management chores of distributed memory allocation, concurrency control, replication and migration control, fault tolerance, and logging and recovery from system and network failures are made available once and for all in the underlying facility VOYAGER, usable by many different data-intensive applications.

In addition to the implementation of the VOYAGER prototype system, we continue to tackle technical issues in the design, analysis and validation of the algorithms used in this facility in order to understand the potential performance bottlenecks. Application testbeds are also included in the investigation so as to better understand the nature of the types of data-intensive parallel applications that can best take advantage of the services provided by the distributed shared virtual memory facility.