Simplifying the Programmer's Job

Indeed, if minimizing latency while allowing scale were the only goal of the server developer, that developer would be best served by writing his own distributed and multithreaded infrastructure customized for the particular game. But this would require that the server developer deal with the complexities of distributed and concurrent programming. Before getting too obsessed with the need for speed, we should remember that a second, but equally important, goal of Darkstar is to allow the production of multithreaded, distributed games while providing the programmer a model of writing on a single machine in a single thread.

To a considerable extent, we have succeeded in this goal. By wrapping all tasks in transactions and detecting data conflicts within the Data Service, programmers get the benefits of multiple threads without needing to introduce locking protocols, synchronization, or semaphores into their code. Programmers do not have to worry about how to move a player from one server to another, since Darkstar handles the load balancing transparently for them. The programming model, although stylized and restrictive, has been found by early members of the community to be natural for the kinds of games and virtual worlds that they are building.

Unfortunately, we have found that we can't hide everything from the programmer. This became apparent when the very first game to be written on top of Darkstar showed very little parallelism (and exceptionally poor performance). On examination of the source code, it did not take us long to find the explanation. The data structures in the game had been written in such a way that any change of state in the game involved a single object, which was used as a coordinator for everything. The use of this single object effectively serialized all of the actions within the game, making it impossible for the infrastructure to find or exploit any concurrency.

Once we saw this, we had a long discussion with the game developers about the need to design their objects with concurrent access in mind. An audit of the data objects in the game showed a number of similar cases where concurrency was (unintentionally) precluded by choices made in the data design. Once these objects were redesigned, the performance of the overall system increased by multiple orders of magnitude.

This taught us that it is not possible for the developers using Darkstar to be completely ignorant of the underlying concurrent and distributed nature of the system. However, their knowledge of these properties of the system need not include the usual problems of concurrency control, locking, and dealing with communication between the distributed parts of the system. Instead, they are confined to the design activity of ensuring that their data objects are defined in such a way that concurrency can be maximized. Such design usually takes the more general form of ensuring that the objects defined are self-contained and do not depend on the state of other objects for their own operations, which is not a bad design principle in any system.