Task Portability

The core of the ability to balance load is that, given the programming model we require and the basic stack services that must be used, tasks that are performed in response to a client-generated or game-internal event are portable from any of the machines running a copy of the game or world logic on a Darkstar stack to any other machine running such a copy. The tasks themselves are written in Java,† which means that they can be run on any of the other machines as long as those (physical) machines have the same Java Virtual Machine as part of the runtime stack. All data read and manipulated by the task must be obtained from the Data Service, which is shared by all of the instances of the game or virtual world and the Darkstar stack on all of the machines. Communication is mediated by the Session Service or by Channels, which abstract the actual endpoints of the communication and allow any particular session or channel to be moved from one server to another. Thus, any task can be run on any of the instances of the game server without changing the semantics of the task.

This makes the basic scaling mechanism of Darkstar seemingly simple. If there is a machine that is being overloaded, simply move some of the tasks from that machine to one that is less loaded. If all of the machines are being overloaded, add a new machine to the group running a copy of the game or virtual world server logic on top of a Darkstar stack, and the underlying load-balancing software will start distributing load to that new machine.

The monitoring of the load on the individual machines and the redistribution of the load when needed is the job of the meta-services. These are network-level services that are not visible to the game or virtual world programmer, but are seen by and can themselves observe the services in the Darkstar stack. These meta-services observe, for example, which machines are currently running (and if any of those machines fail), what users are associated with the tasks on a particular machine, and the current load on the different machines. Since the meta-services are not visible to the game or virtual world programmer, they can be changed at any time without having an impact on the correctness of the game logic. This allows us to experiment with different strategies and approaches to dynamically load balance the system, and allows us to enrich the set of meta-services as required by the infrastructure.

The same mechanism that we have used for scaling over multiple machines is used to obtain a high degree of fault-tolerance in the system. Given the machine-independent nature of the data that is used by a task and the communication mechanisms, it may be clear that it is possible to move a task from one machine to another. But if a machine fails, how can we recover the tasks that were on that machine? The answer is that the tasks themselves are persistent objects, stored in the Data Service for the overall system. Thus, if a machine fails, any of the tasks that were being performed by that machine will be treated as aborted transactions, and will be

† More precisely, all of the tasks consist of sequences of bytecodes that can be executed on the Java Virtual Machine. We don't care what the source-level language is; all we care about is that the compiled form of that source language can be run on any of the environments that make up the distributed set of machines running the game or virtual world.