Fly Object Space

On A Page

It is becoming increasingly necessary to build highly concurrent software systems such as those found on grids, clusters, and groups of multi-core processing systems. However as the number of systems and processing units increases, the complexity of software to manage and coordinate the operation of these systems also seems to grow exponentially. Fly Object Spaces are aimed directly at this problem.

Stemming from research conducted at Yale University by David Gelernter and Nicholas Carriero, an extremely elegant solution to the problems of distributing and locking objects (state) in a system has been in use in various forms for over 20 years - the 'Space' based model of distributed computing. This has been embodied in a number of Space implementations, normally named for the software language in which they are implemented – for example LindaSpaces for the Linda language, JavaSpaces for the Java™ language, and Rinda (after Linda) for the Ruby Language.

The Fly Object Server extracts the fundamental object based operations required by all of these systems and implements them as a binary executable to run on a host operating system, in the form of a software service. This means that Fly based systems provide much higher throughput and scalability than has been previously possible with systems based around each of the hosting languages. The development of Fly has the promise to make a raft of systems based on the Spaces architecture practical, including those in financial trading and modelling, video and audio processing, as well as bio-informatics applications such as protein folding and structure prediction for drugs development programmes.

The Fly Server must be bound back into a host language environment so that it can transmit objects created in that language environment. At the time of writing the Server has Java, Scala and Ruby language bindings. Although JavaSpaces are the most prevalent of all Space based systems, we have seen great interest in the Scala interface and the use of Scala Actors .

Although Fly has a lighter and less complex set of interfaces than the JavaSpaces interfaces, the core space operations (Write, Read and Take) implement directly comparable behaviour. In tests, the scalability and throughput of the Fly server far exceeds any comparable Java Space implementation without the pauses caused by garbage collection, which can reduce latency for highly concurrent applications.

Tags : Object Spaces, Cloud, Cluster, Parallel, Complexity, Distributed Locking, Throughput, Language Neutral, Distributed Coordination, Order Routing, Monte Carlo, Modelling.