Specus - Opensource Minecraft server

Tomas Herman

2012-01-24 Tue

Contents

1 Motivation **2**
- 1.1 What is Minecraft . . . 2
- 1.2 Why another Minecraft server . . . 2

2 Goals **2**

3 Tools **3**
- 3.1 Scala language . . . 3
- 3.2 Redis database . . . 3
- 3.3 Actor model . . . 3
- 3.4 Akka library . . . 3
- 3.5 Other tools . . . 3

4 Implementation details **4**
- 4.1 Distributabilty . . . 4
- 4.2 Extensibility . . . 4
- 4.3 Stats plugin . . . 5
  - 4.3.1 Http frontend plugin . . . 5
  - 4.3.2 Minecraft plugin . . . 5

5 Conclusion **5**

1 Motivation

1.1 What is Minecraft

There has been a lot of buzz lately about very well received indie game called Minecraft. In this game, players are presented with semi-randomly generated worlds made of blocks. In these worlds, players can do whatever they want. Some choose to go explore randomly generated cave systems in hopes to find rare diamond blocks, others build huge cities, rollercoaster or even USS Enterprise space ships. Building these types of huge objects takes a lot of time and effort. Luckily, Minecraft supports mulitplayer games.

1.2 Why another Minecraft server

There is a couple of Minecraft server implementations. Obviously, there is the offical one, made by the creator of Minecraft. However, there are some problems with that one. The main problem is, that there is no way to extend it. The server is distributed in obfuscated form, so it takes a lot of effort to figure out how to alter the server behaviour (it is possible, however, and it has been done).

Another drawback of official implementation is the poor performance. The official server uses threaded model for network concurrency, rather then multiplexing, which is a huge problem if one tries to build a trully scalable server. It also uses disk memory as persistence for data, which is good in order to keep the RAM requirements lower, but introduces more latency.

Because of all the above mentioned problems, i think it is reasonable to try and build a better version.

2 Goals

The goals of this project are as follows:

- Build a Minecraft server that will implement part of the Minecraft functionality (creative mode)
- Use TCP multiplexing
- Use in-memory persistence database
- Use Actor model for concurrency

3 Tools

3.1 Scala language

Specus is written in Scala. Scala is a language that tries unify functional and object oriented programming constructs. For example, Scala features include first class functions, higher order functions, persistent datastructures, type inference aswell as object oriented features such as mix-ins.

Scala also runs on JVM, which means that it can make use of all the great JVM features, such as inlining and JIT compilation, thus making it great candidate for server side platform.

3.2 Redis database

For presistencem, i will be using Redis database. It's an eventually-consistent high performance server that allows users to store key-value pairs. Unlike other key-value pairs, though, it allows values to be either an list, hash map, set, binary string, or ordinary value.

3.3 Actor model

Actor model is a model developed for programming telephone network switches. It is a way to create scalable, fault tolerant and highly avalible services. The idea is, that a service is made of many sub services which communicate among each other by sending messages. Those services are also linked into tree like topologies. Then, when a service fails, a message is sent to service that was linked with the failing service which the receiving service can use to coordinate the recovery of failing service.

3.4 Akka library

Akka is a middleware library providing many great features for Scala language. Other than providing actor system described above, it provides other very useful services for concurrent software, such as Software transactional memory and Futures.

3.5 Other tools

Other tools used in this project are:

- netty for simpler network IO

- jetty and scalatra for http frontend plugin
- Specs2 for unit testing

4 Implementation details

4.1 Distributabilty

Specus use proxy architecture to deal with distributivity. It uses a single JVM to handle all conections from clients. It decodes sent data into Scala classes and then sends those classes into nodes connected to it. The nodes then process the message. Messages are delivered to random nodes (round robin in practice), so all the stateful information need to be stored inside the Redis database.

4.2 Extensibility

Extensibility was the most important goal when designing the specus. Specus project itself is split into 5 modules

- server api - api that is implemented in server and can be used in plugins
- node api - api that is implemented in node and can be used in plugins
- server api - implementation of server api and server functionality
- node - implementation of node api and node functionality
- common api - api shared among server api and node api

Basically, only hardcoded functionality in server and node is parsing of tcp stream into messages (defined via plugin), plugin loading and plugin communication and communication between server and nodes. Everything else is provided via plugins. There is a mechanism inside Specus that allows plugins register for different kind of messages and Specus itself sends those messages if there is anything interesting happening (client connected, client disconnected etc).

Plugins are basically just JAR files that contain JSON file in a specific directory that describe plugin. It can contain information about author, it can provide plugin dependencies to be checked on server startup and it provides a name of class describing more information about the plugin.

In Specus sytem, the functionality provided by plugin looks like a normal actor service. It can send and receive messages and is responsible for handling all failures inside itself.

There are 3 plugins implemented as proof-of-concept.

4.3 Stats plugin

Stats plugin is a very simple plugin that gathers messages from server and stores them. For example, it keeps track of how many clients are connected to the server. It also keeps track of all the messages send from and to a connection, which may be useful for debugging. It also provides messages that allow other plugins to request these data.

4.3.1 Http frontend plugin

This plugin contains a Jetty server and allows other users to connect to it via http protocol. It's basically a front end for Stats plugin. It requires the plugin to be avalible and it will fail without it.

4.3.2 Minecraft plugin

And of course the Minecraft plugin, implementing the entire Minecraft functionality. The server part consists of message definitions (data to be parsed from tcp stream from client) and codecs providing encoding and decoding functionality.

The node part consists of multiple actor services that handle messages. These processors have avalible connection to the redis presistence inside which they keep all the state of the game.

5 Conclusion

In conclusion, using scala and especially Akka library was a real pleasure. The combination of immutable data structures and first class functions made it very easy to implement some rather complicated stuff. For example, when parsing TCP streams, i was able to come up with codec implementation that would figure out how to parse message just from it's definition (= it's class), which saved me a lot of time. I still had to write a few parsers by hand for the more complicated messages.

The Akka library proved to be superb. It is very easy to do asynchronous messaging inside a system thanks to it's Futures and Actors.