Debugging your *MAvis* client

Lasse Dissing Hansen and Thomas Bolander

February 2, 2025

Debugging a *MAvis* client is slightly more complicated than the usual "press Debug in your IDE" workflow since the server starts the client as a sub-process. Trying to start the server from the IDE won't help since this results in the debugger attaching to the server process rather than the client. Instead, the server should start the client in a special debug mode which allows us to attach a debugger afterwards. There are many ways to accomplish this, e.g., using a wrapper script or named pipes, but we'll focus on remote debugging since it is supported out of the box in most modern toolchains and development environments. It's a feature which is intended for scenarios where it is impractical to develop on the same machine as the program runs on, e.g. in embedded or high-performance computing but it is equally useful for this purpose. The remote debugging workflow is as follows: 1) Open your client in the IDE, 2) Configure your client program for remote debugging in the IDE, 3) start the server which then starts your client, and 4) attach your debugging to the client where you can then inspect and step through it as usual. The exact steps needed depend on what language and development environment you are using, but we'll demonstrate it using Java.

1 Java

First open your client in the IDE. For instance, in IntelliJ this would be by selecting `File -> Open...` and then choosing the folder with your client source files. If you have been compiling from the command line up till now, make sure to delete your class files first, so that you open a folder with only the source files in your IDE. Compile the project. Navigate to the parent directory of the directory containing the compiled files. For instance, in IntelliJ, this is by default the subdirectory `out/production` of the directory of your source files. Usually, we would run the client as follows, from the parent directory of the directory containing the client source files:

```
java -jar server.jar -l levels/SAD1.lvl -c "java searchclient.SearchClient"  -g
```

However, now we are in another directory, so make sure to update the paths:

```
java -jar <path-to-server>/server.jar -l <path-to-levels>/SAD1.lvl
     -c "java searchclient.SearchClient"  -g
```

We can now configure the JVM to start the client in remote debugging mode by adding a couple of parameters to the client option:

```
java -jar <path-to-server>/server.jar -l <path-to-levels>/SAD1.lvl
     -c "java -agentlib:jdwp=transport=dt_socket,server=y,
     address=8000,quiet=y,suspend=y searchclient.SearchClient" -g
```

This will make the JVM start the program in a paused state and open a debugging server at `localhost:8000`. The next step is to attach your debugger to the paused client. The details will depend on what IDE you are using but in all cases you will need to configure the debugger to attach to a remote program running at `localhost:8000`. As an example we can use the debugger directly from the command line:

```
jdb -attach localhost:8000
```

or in your favourite IDE:

- IntelliJ Idea
  `https://www.jetbrains.com/help/idea/attaching-to-local-process.html#attach-to-local`

- Eclipse
  https://techhub.hpe.com/eginfolib/networking/docs/sdn/sdnc2_7/5200-0910prog/content/s_sdnc-debug-with-eclipse.html

2 Python

The standard Python debugger `pdb` doesn't support remote debugging out of the box and we will therefore have to rely on third-party libraries to achieve the same effect. In the following we'll demonstrate how to setup VS Code with remote debugging using the `debugpy` library. First, install the library using pip (or however you prefer):

```
pip install --user debugpy
```

We can then use this library to setup a debug server from within our application, e.g., by adding the following code snippet to the start of your *MAvis* client:

```
import debugpy

debugpy.listen(("localhost", 1234)) # Open a debugging server at localhost:1234
debugpy.wait_for_client()           # Wait for the debugger to connect
debugpy.breakpoint()                # Ensure the program starts paused
```

Next step is then to create a VS Code launch configuration which connects to the debugpy server started by your *MAvis* client. One way to do this is to open the text file `.vscode/launch.json` in your project folder (or create the file if it doesn't exist), and add a new configuration, e.g.,

```
{
  "version": "0.2.0",
  "configurations": [
      {
          "name": "MAvis",
          "type": "python",
          "request": "attach",
          "connect": {
              "host": "localhost",
              "port": 1234
          },
          "pathMappings": [
              {
                  "localRoot": "${workspaceFolder}",
                  "remoteRoot": "."
              }
          ],
          "justMyCode": true
      }
  ]
}
```

The `pathMappings` property tells the debugger where it can find the source code for your program and the `justMyCode` property instructs the debugger to not step into library function calls. With this added, you should be able to select a "MAvis" configuration in the Run and Debug tab on the VS Code activity bar. Starting the debugger is then as simple as 1) Starting the *MAvis* server with your client as usual and 2) press "Start Debugging" in VS Code. A similar setup is also possible for PyCharm as described in:
https://www.jetbrains.com/help/pycharm/remote-debugging-with-product.html#remote-debug-config

3 Other languages

For those of you who aren't using Java or Python, we have compiled a list of links for some of the more common languages and toolchains, but even if your setup is not mentioned here, you should be able to find a guide by simply Googling "remote debugging `$YOUR_FAVORITE_LANG`"

C, C++, Rust, Zig and a ton of other natively compiled languages

- *LLVM (Windows, OS X and Linux)*
  `https://lldb.llvm.org/use/remote.html`
- *GNU Compiler Collection (Linux)*
  `https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gnat_ugn_unw/Remote-Debugging-using-gdbserver.html`
- *Visual Studio (Windows)*
  `https://learn.microsoft.com/en-us/visualstudio/debugger/remote-debugging-cpp?view=vs-2022`

C# (and probably also F#)

`https://learn.microsoft.com/en-us/visualstudio/debugger/remote-debugging-csharp?view=vs-2022`