Task: Write a paper comparing the similarities and differences between SpiderMonkey and V8

SpiderMonkey

SpiderMonkey is the code name for the first **JavaScript engine**, written by Brendan Eich at Netscape Communications. Later, it is released as open source and currently maintained by the Mozilla Foundation. SpiderMonkey is the JavaScript engine used in the Firefox web browser. It is written in C/C++ and contains an interpreter, the IonMonkey JIT compiler, and a garbage collector. It implements the ECMA-262 clientside scripting interoperability of web pages.

Warp, a significant update to SpiderMonkey in Firefox 83 has improved JavaScript performance. With Warp (also called WarpBuilder), we're making big changes to our JIT (just-in-time) compilers, resulting in improved responsiveness, faster page loads and better memory usage. The new architecture is also more maintainable and unlocks additional SpiderMonkey improvements.

Multiple JITs

The first step when running JavaScript is to parse the source code into **bytecode**, a lower-level representation. Bytecode can be executed immediately using an interpreter or can be compiled to native code by a just-in-time (JIT) compiler. Modern JavaScript engines have multiple tiered execution engines.

JS functions may switch between tiers depending on the expected benefit of switching:

- **Interpreters and baseline JITs** have fast compilation times, perform only basic code optimizations (typically based on Inline Caches), and collect profiling data.
- The **Optimizing JIT** performs advanced compiler optimizations but has slower compilation times and uses more memory, so is only used for functions that are warm (called many times).

The optimizing JIT makes assumptions based on the profiling data collected by the other tiers. If these assumptions turn out to be wrong, the optimized code is discarded. When this happens the function resumes execution in the baseline tiers and has to warm-up again (this is called a *bailout*).

Profiling data

Our previous optimizing JIT, Ion, used two very different systems for gathering profiling information to guide JIT optimizations. The first is Type Inference (TI), which collects global information about the types of objects used in the JS code. The second is CacheIR, a simple linear bytecode format used by the Baseline Interpreter and the Baseline JIT as the fundamental optimization primitive. Ion mostly relied on TI, but occasionally used CacheIR information when TI data was unavailable. With Warp, we've changed our optimizing JIT to rely solely on CacheIR data collected by the baseline tiers.

There's a lot of information here, but the thing to note is that we've replaced the IonBuilder frontend (outlined in red) with the simpler WarpBuilder frontend (outlined in green). IonBuilder and WarpBuilder both produce Ion MIR, an intermediate representation used by the optimizing JIT backend.

Where IonBuilder used TI data gathered from the whole engine to generate MIR, WarpBuilder generates MIR using the same CacheIR that the Baseline Interpreter and Baseline JIT use to generate Inline Caches (ICs). As we'll see below, the tighter integration between Warp and the lower tiers has several advantages.

Google Chrome V8

V8 is Google's open source high-performance JavaScript and Web Assembly engine, written in C++. It is used in Chrome and in Node.js, among others. It implements ECMAScript and WebAssembly, and runs on Windows 7 or later, macOS 10.12+, and Linux systems that use x64, IA-32, ARM, or MIPS processors. V8 can run standalone or can be embedded into any C++ application.

The V8 engine has an interpreter named "Ignition" [5]. This interpreter is used for interpreting and executing low level bytecode. Bytecodes, although slower, are smaller than machine codes and requires lesser compilation time.

To compile Javascript to bytecode, the Javascript code has to be parsed to generate its Abstract Syntax Tree (AST). The interpreter has an accumulator register, which allows it to reduce the size of the bytecode. The overall design makes Ignition a highly efficient interpreter.

Whereas its optimizing JIT compiler is named "TurboFan" [6]. TurboFan will profile the code and see if it is used multiple times throughout the entire Javascript execution. If it is, the code will be dynamically optimized immediately into machine code, without any intermediate binary code. If it is a one-time executed "non-hot" code, it will only be compiled into binary code.

By reducing unnecessary generation of machine code, the Javascript engine will be able to run more efficiently. The profiling makes use of hidden classes, which are classes that can be instantiated to create objects with fixed variable offsets. Fixed offsets, rather than dynamic lookup, allows codes to be read in a very efficient manner without having to resolve to a memory location for a variable.

However, if the code being profiled is acting in a way which is not as predicted, the engine will fallback to normal bytecode interpretation, and this causes it to slow down. Only after some period will V8 then attempt to profile other codes. Therefore, developers should always try to write their algorithms and code that runs in a predictable manner.

Garbage collection is also done in a "stop-the-world", generational way. This means that before the JavaScript engine does garbage collection, all processing of JavaScript will be paused, and the garbage collector will find objects and data that are no longer referenced and collect them. This ensures that garbage collection is done in an accurate and efficient way.

V8 is the fastest because it compiles all JS to machine code. SpiderMonkey (what FF uses) is fast too, but compiles to an intermediate bytecode, not machine code. That is the major difference with V8.

Differences between V8 and SpiderMonkey

S. No	V8	SpiderMonkey
1.	Relatively new and first introduce as part of google chrome.	First written by Brendan Eich, released as open source and currently maintained by Mozilla FF.
2.	V8 eliminates the need for interpreter by compiling JS to machine instructions.	Compiles JS code to an intermediate interpreted language
3.	V8 is the fastest in compiling JS code into machine code.	SpiderMonkey is also fast, but slower than V8 particularly for complex code.

4.	V8 is embedded in such server-side technologies, as Node.js, MongoDB, and Couchbase.	SpiderMonkey is embedded in Mozilla Firefox or other applications.
The biggest difference is in compilation. SpiderMonkey compiles JavaScript to an intermediate language which is interpreted. V8 differs by compiling JavaScript to machine instructions, eliminating a need for an interpreter.		

Similarities between V8 and SpiderMonkey

Both are JavaScript engines intended to parse, compile and interpret JavaScript code in to machine readable byte code during the same time parallel to execution of the code. With minor differences in employed steps, JavaScript code passes through lexical analysis to tokenize the code and build Abstract Syntax Tree to translate to Byte code thereby execute efficiently.

References:

https://en.wikipedia.org/wiki/SpiderMonkey

https://hacks.mozilla.org/2020/11/warp-improved-js-performance-in-firefox-83/

https://dzone.com/articles/v8-javascript-engine-the-non-stop-improvement

https://hacks.mozilla.org/2020/11/warp-improved-js-performance-in-firefox-83/

https://developers.redhat.com/blog/2016/05/31/javascript-engine-performance-comparison-v8-charkra-chakra-core-2#javascript_engines