Analysis and Performance Evaluation of HHVM

Xuan Wang Chutian Shen

Abstract

The HipHop Virtual Machine(HHVM) is a Just-In-Time(JIT) compiler and runtime for PHP and the Hack language(Hacklang). HHVM converts PHP to bytecode. In PHP programs, some variables can be polymorphic variables, and need proper handling from the runtime system. This requires the runtime system to reduce the total number of type checks to prevent catastrophic slowdown.

HHVM has the concept of a *tracelet*. A tracelet is approximately a small portion of code with type guards. Through tracelets, HHVM can learn the types of programs by observation, and it can optimize tracelets to improve efficiency.

In this project, we study the design considerations of HHVM, evaluate performance of PHP 5.6 and PHP 7.0 interpreters with HHVM by using microbenchmarks and real-world open-source PHP applications. We also compare the performance of HHVM with Go, Java(HotSpot) and Python(PyPy). Moreover, we build a web application similar to Go Playground; with this application, a user can run PHP or Hacklang code on a web page.

1. Introduction

PHP is a widely-used programming language for scripting and building websites. Even the biggest social media and social networking service company facebook was developed in PHP. However, PHP has difficulty in handling polymorphic variables and expressions caused by PHP's dynamically typed values natural. This disadvantage makes PHP not scalable enough. Therefore, Facebook developed a more efficient runtime system, HHVM, for PHP programs.

Along with HHVM, Facebook introduced a new programming language--the Hack language, or Hacklang. Hacklang allows developers to annotate data types for variables, and this can further improve JIT compiler's efficiency.

2. HipHop Virtual Machine

Although by means of easy deployment, excellent documentation, PHP becomes one of the most popular programming languages in web application development, PHP applications do not run efficiently enough due to late-bound, dynamically type properties. The current PHP runtime system implementation is interpreter-based. The dynamic typed nature of PHP significantly slows down the performance. All variables in PHP user program are the same union type, and the PHP interpreter need to check types constantly. Moreover, the "array" type in PHP bears both array list and hash table semantics, which is not an efficient approach.

Before HHVM, there is a compiler, HipHop compiler[1], which is also developed by Facebook. This compiler can greatly improved the performance of PHP applications. HipHop compiler compiles PHP into C++ and then produce a native binary. However, this approach doesn't favor developers' productivity, because the compilation time is long, and the compiled binary is much larger than PHP code for a web site at Facebook's scale.

HHVM is a stack-based virtual machine. The Ahead-of-Time compiler will convert PHP code and

Hacklang code into HipHop Byte Code (HHBC). Each HHBC is encoded using one or more bytes, where the first byte is the opcode and others are immediate arguments. Temporary values are operated on the evaluation stack. HHVM is designed as two parts: the frontend Ahead-of-Time compiler and the backend Just-In-Time compiler, HHBC is the decoupling boundary between its frontend and backend. The frontend converts PHP into HHBC. The backend execute HHBC without touching the original PHP codes[2].

HHVM is compatible with the PHP language, and most PHP programs and libraries can run on HHVM. It also allows PHP code and Hacklang code run together. This is because HHVM will convert PHP code and Hacklang code into HHBC, and map HHBC to machine code.

HHVM's JIT compiler is the major contributor for performance improvement. This JIT compiler is responsible for identifying types and using this type information to generate efficient machine code.

In statically typed languages, types are bound to variables, and in that case the values can be represented without tags, the operations on the value can be statically bound. However, in dynamic languages, such as PHP and Python, ahead-of-time type inference and binding are generally unavailable. Considering the following snippet, the value of $foo cannot be determined until runtime.

```
$foo = bar() ? 1024 : "overwhelming";
```

However, for a real world program, the number of possible types of a variable is usually limited. It is very rare for a variable to be assigned and reassigned with multiple different types, because it will make the code difficult to maintain in a production environment. For example, in the above mentioned snippet, although $foo has dynamic types, it can only be either an integer or an string.

With this observation, HHVM's JIT compiler handles dynamic typed programs by compiling the code at the granularity of tracelets, while other JIT compiler usually compiles whole files, methods, or traces. A tracelet is a single-entry, multiple-exit region of the source program[1], and a tracelet is usually smaller than a basic block. In a tracelet, the type of variables flowing into the region are annotated. HHVM annotates each HHBC instruction with input types and output types.

At runtime, if the input types of a tracelet were not annotated, HHVM would observe the program's runtime state at JIT-compile time and annotate the observed types. And afterward, HHVM will use these observed types as prediction for future execution of the tracelet. Comparing to statistical value-profiling approaches[3], this approach is less expensive and doesn't require long warm-up time.

When generating the corresponding machine code for a tracelet, HHVM's JIT first check the types at the entrance to ensure that the type conditions of the tracelet hold at run time. If the type checking(guard) pass, the JIT compiler can do multiple optimization for the tracelet body, since all the type information is observed and definite within the tracelet body, therefore, the machine code is type specified.

When type checks at the entrance of a tracelet fails, the JIT compiler will kick in to compile the tracelet into a new version. In this new version, the JIT compiler will put new type guards and annotate new types on HHBC based on the new observed types and compile the tracelet into machine code with the new type information. The HHBC with new type guards will be chained after the previous version, such that when the guards in the previous tracelet fails, HHVM will try to execute this new version of the tracelet in the future at run time. At the same time, the compiled machine code will stay in code cache, as to improve the performance when the tracelet being

executed in the future.

In a more polymorphic situation, a variable can have a variety of possible types. In this case, HHVM will chain a series of type annotated HHBC and perform a linear search for the corresponding types. If the type checking from all the annotated tracelet HHBC all fails, the JIT compiler will annotate the HHBC with observed types and generate machine code. In the design of HHVM, when a chain of 12 type-guarded versions of a tracelet all fails, HHVM will use a bytecode interpreter to execute the program. This situation rarely happens, and in this case, continuing chaining more type-guarded tracelet won't improve the performance significantly, because the time spent on linear search for the correct types is already comparable with the execution time using an interpreter.

The intermediate representation of HHBC is called HHIR. The HHVM JIT compiles a tracelet's HHBC into HHIR, representing the tracelet's HHBC instructions and type information. Unlike HHBC, this intermediate representation is a typed representation, which enables the JIT compiler to map the HHIR to machine code with more optimizations because the type information is definite. The HHIR is also machine independent but designed as close to a general architecture as possible. This reduces the complexity and effort to implement HHVM for different architectures, while it enables HHVM to produce efficient code at the same time.

In general, the design of HHVM enables compilation and optimization of a tracelet with fully disambiguated types. In this way, it only moderately degrade the performance in a highly polymorphic situation. At the same time, the JIT compiler reduces the recompile overhead by dividing the whole program into tracelets, which is a smaller granularity.

At the same time, because HHVM only has a small JIT scope, tracelet, which might be much smaller than a method or a file, the global optimization of the JIT compiler is missing. Therefore, the overall performance improvement may not be significant in certain cases. Moreover, the frequent insertion of type guards may affect the performance, in the case of small tracelets, because linearly searching for the correct type annotated HHBC block may be slower than interpretation in certain cases. Also, because a tracelet is much smaller than a method, HHVM cannot harness the existing compiler infrastructures like LLVM and the performance benefits out of it.

3. The Hack Language

HHVM comes with a new programming language, Hacklang. Hacklang supports type annotation and a mixed type system. A developer can partially annotate types in Hacklang code, and that enables maximum interoperability with PHP code. The developer can also choose to fully annotate types on the Hacklang code; this enables maximum type coverage and there can be no type errors at runtime. This approach also improves performance of HHVM, because all the variables are annotated with types, thus there will be only one version of the type-guarded HHBC generated by the JIT compiler, instead of a chain of type annotated tracelets, as mentioned in the previous section. Therefore, the number of type checking at run time can be significantly reduced. From a software engineering perspective, this design allows developers to gradually convert PHP code to type annotated Hacklang code, while keeping PHP code and Hacklang code together in production. Facebook also promotes Hacklang as a general programming language, and now HHVM supports both x86-64 and ARMv8 architectures.

The same engineering practice for recent programming languages is the interoperability between Java and Kotlin. Kotlin code will be coverted into JVM byte code. And there are even tools for

converting Java program into Kotlin. However, this kind of tools don't exist in the PHP world, because PHP is a dynamic language, while Java and Kotlin are both statically typed, and the tools can one-to-one map Java types to Kotlin ones. We can expect new languages and runtime systems will probably follow this interoperable approach, especially in the industry.

The language design of Hacklang incorporates many modern language features. In Hacklang, explicit claim of nullable types is enforced to prevent potential null pointer exceptions. Hacklang also supports generics and lambda expressions. Hacklang also has collections with different implementation from PHP. Unlike PHP's array, which bears both hashtable and arraylist semantics, each collection in Hacklang only has one set of interface. In general, the language design of Hacklang is similar to a general modern object-oriented language, and we can see many features comes from Java. This also enables Java developers learn Hacklang and become productive very quickly.

4. Performance Evaluation

In our performance evaluation, we first used some of the popular benchmarks and constructed our own benchmarks to test the speedups of HHVM over PHP Zend interpreters. For the versions of PHP interpreters, we used PHP 5.6.30 and PHP 7.0.18, because these two versions of PHP runtime are the two mostly used PHP runtimes, the previous generation and the current generation. The HHVM we used is version 3.18, which is the latest stable release of HHVM in 2017. The system is an AWS EC2 t2.micro instance, which has 1 virtual CPU up to 3.3 GHz and 1.0 GB memory.

In our benchmark test, we report the performance of each runtime system in terms of its speedup value over PHP 5.6. We choose 10 benchmarks. The result for PHP 5.6, PHP 7.0, and HHVM 3.18 are summarized in Figure 1. HHVM achieves a speedup of 3.8x over PHP 7.0 and a speedup of 9.3x over the PHP 5.6 interpreter. We found that for benchmarks with large number of recursive or repetitive function calls, HHVM outperform PHP interpreters. However, for benchmarks involving large-scale computation and written in top level scope or the functions being executed only once, HHVM doesn't show superiority in terms of speedups. The benchmark *binarytrees* is a benchmark for garbage collection, and we can see HHVM achieves a speedup of 7.2x over PHP 5.6, and 2.88x over PHP 7.0.

To test the compatibility and performance of HHVM, we evaluated two of the most popular open-source PHP applications--MediaWiki and WordPress. MediaWiki is basically an open-source version of Wikipedia, and many organizations use it to create their own wiki pages. WordPress is a popular PHP framework for building blog websites. We set a nginx server for each open-source project on an AWS t2.micro instance, and test the round-trip http request time for PHP5.6, PHP 7.0, and HHVM 3.18.

As shown in Figure 2, HHVM achieves 1.1x to 1.4x speedups over PHP 5.6 and a 1.17x maximum speedup over PHP 7.0. In this case, because the network latency becomes the dominant component, the speedup values are not as significant as in the benchmark tests. However, given the scale of Facebook's billions of users, these speedups can definitely save Facebook a significant amount of CPU time in its datacenters.

We also compared HHVM and other popular languages and runtimes, including PyPy for python, HotSpot for Java and the Go runtime. As shown in Figure 3, HHVM is still slow, comparing with other runtime systems. PyPy has a speedup of 3.3x over HHVM; HotSpot has a speedup of 23.8x over HHVM, and Go runtime has achieved a 19.8x speedup over HHVM. Thus although facebook is

promoting Hacklang as a general programming language and HHVM along with it, the HHVM runtime system is still slower than other popular runtime systems.

Given the JIT compiler of HHVM, if we run a simple hello world program, the total execution time on HHVM is much longer than PHP, because the ahead-of-time compiling time and the initial JIT compilation of tracelets. However, if we repetitively call a function in our program, after enough number of function calls, the start-up time can be amortized. As shown in Figure 4, after enough times of function calls, HHVM outperforms PHP interpreters.

5. HackPlatea: A Web Application

HackPlatea is a web application that receives a PHP or Hacklang program, executes the program, and returns the output. HackPlatea can use PHP 5.6 and PHP 7.0 Zend interpreters, as well as HHVM to execute PHP code. The user can also select HHVM only to execute Hacklang code. Therefore, this application can be used to quickly write some code in PHP or Hacklang without installing all the runtime systems on the user's computer. This application will also give the execution time of each runtime system, and it can be used to quickly evaluate the performance of different runtime systems. Since HHVM uses a JIT compiler and it needs time to start and warm up, when executing simple programs, such as a hello world program, the total execution time of HHVM will be longer than PHP Zend interpreters, as mentioned in the previous section.

In this application, we implemented the web service with Golang as a microservice, because we were working on an alternative implementation of GoPlayground at the beginning of this quarter, and we reused the web service part. The frontend of this application is written in React.js, which is also a Facebook technology. This application runs on Heroku and AWS.

6. Conclusions and Future Work

We learned the language features of Hacklang and the design of HHVM in this project. The Hack language is a big step toward a modern language.

The runtime system for PHP and Hacklang, HHVM, uses an approach of chaining type guarded tracelets to improve its performance. The JIT compiler also improves the performance of the situation that there are repetitive function calls in the program. And for a web service, repetitive calls are very likely to happen.

The next generation PHP(PHPNG) runtime system also uses JIT compiler and support type annotations. We can expect PHPNG will have better performance than PHP 7.0 Zend interpreter, and we look forward to comparisons between PHPNG and HHVM in the future. Moreover, HHVM has many optimizations for production environment. For example, HHVM organizes the HHBC code in a SQL database, which is interesting, and we want to explore more about the reason of this design decision.

References

[1] H. Zhao, I. Procter, M. Yang, et al. The HipHop Compiler for PHP. OOPSLA '12, Oct 19, 2012.

[2] K. Adams, J. Evans, B. Maher, et al. The HipHop Virtual Machine.OOPSLA '14, Oct 20, 2014.

[3] F. Pizlo and G. Barraclough.Value Profiling for Code Optimization, Feb. 13 2014. US Patent App. 13/593,404.

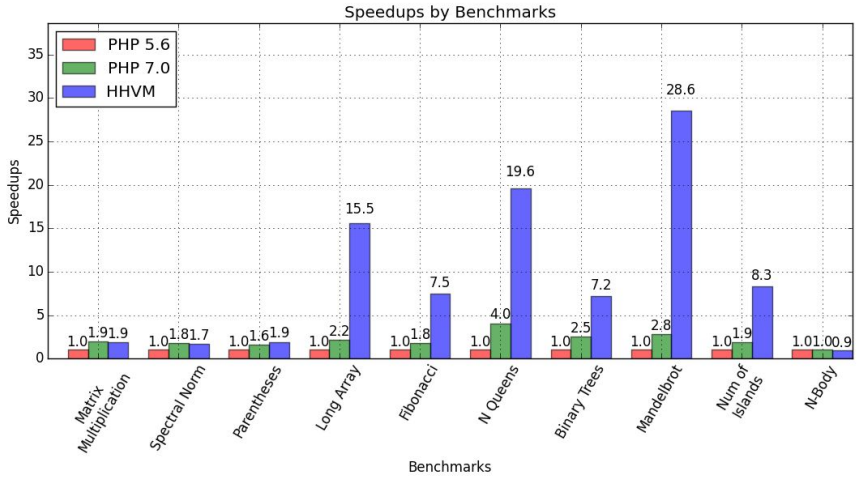
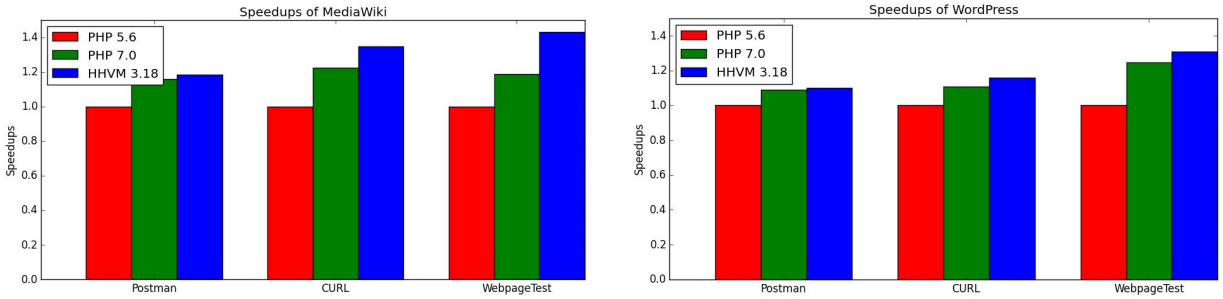


Figure 1. Test results for benchmarks



(a) MediaWiki (b) WordPress

Figure 2.Test results for open-source projects

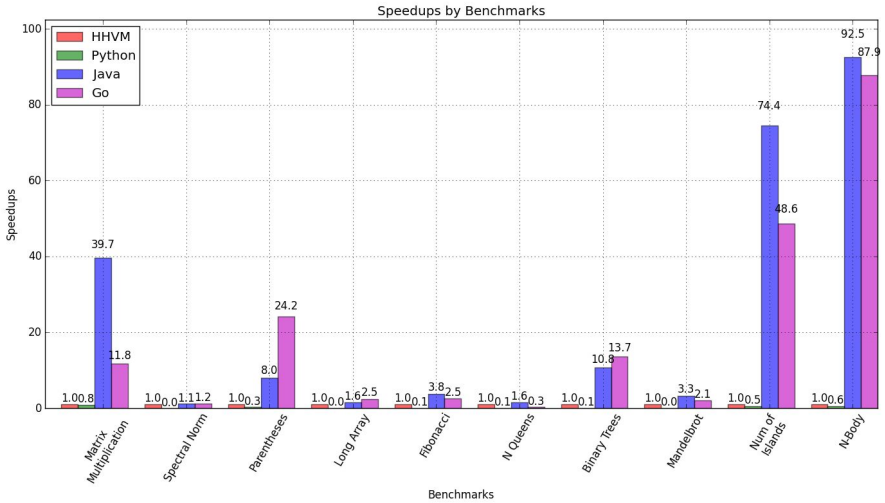


Figure 3. Comparing HHVM, Python, Java, and Go runtimes

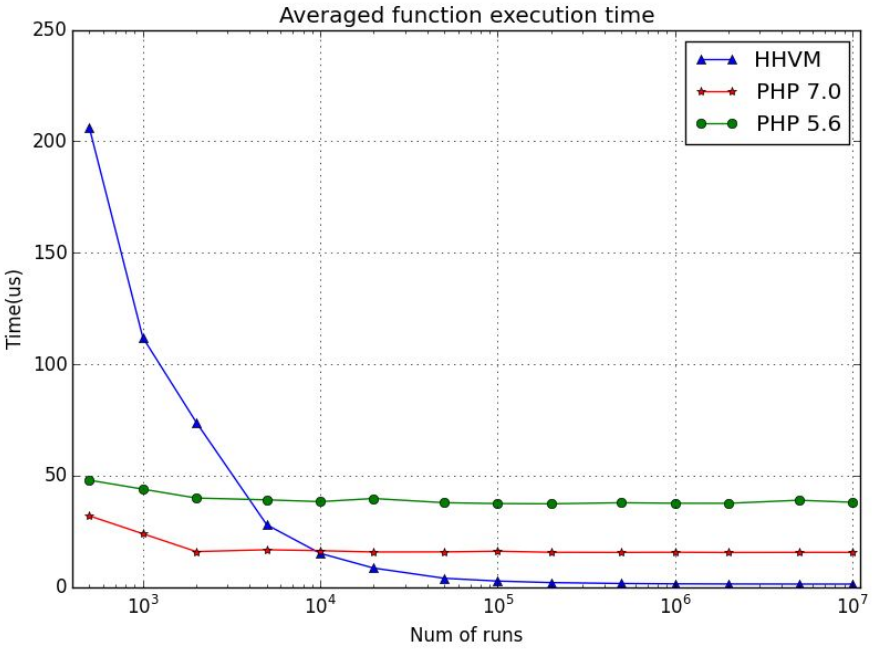


Figure 4. Amortized start-up time