A Limit Study of Thread-Level Speculation in JavaScript Engines — Initial Results

Jan Kasper Martinsen and Håkan Grahn
School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
{Jan.Kasper.Martinsen,Hakan.Grahn}@bth.se

Anders Isberg
Sony Mobile Communication AB
SE-221 88 Lund, Sweden
Anders.Isberg@sonymobile.com

ABSTRACT

JavaScript is a programming language for interactive clientside functionalities in web applications. It is a sequential programming language, so it cannot take advantage of multicore processors. Previously Thread-Level Speculation has been used to take advantage of multicore processors for JavaScript execution in web applications with promising results execution time wise, but with a large memory overhead. In this study we have evaluated the effects of limiting the amount of memory, the number of threads and the depth of speculation in Thread-Level Speculation. Our results indicate that we can tune these parameters to improve execution time and reduce the memory overhead.

1. INTRODUCTION

JavaScript is a dynamically typed, object based scripting language with run-time evaluation typically used for dynamic clientside functionality in web applications. Several optimization techniques have been suggested to decrease the execution time [4, 14, 8]. However, the decrease in execution time has been measured on a set of benchmarks, which have been reported unrepresentative for JavaScript execution in real-world web applications [5, 11, 12]. A result of this is that optimization techniques such as just-in-time compilation often fails to decrease the JavaScript execution time in popular web applications [6].

Fortuna et al. [3] showed that there is a significant potential for parallelism in many web applications with a potential of a speedup up to 45 times compared to the sequential execution time. To take advantage of this observation and to hide the complexity of parallel programming from the JavaScript programmer, one approach is to dynamically extract parallelism from a sequential program using Thread-Level Speculation (TLS) [13]. The performance potential of TLS has been shown for applications with static loops, statically typed languages, and in Java bytecode environments [9, 10]. In [7], we used the Squirrelfish JavaScript engine, which is part of WebKit, and performed experiments on a number of popular web applications.We employed an aggressive speculation scheme, so even if we were able to decrease the execution time significantly, our approach had a high memory overhead.

In this paper we extend previous studies by limiting the execution resources for TLS, i.e., the amount of memory, the number of threads, and the speculation depth. We implement the limitations in Squirrelfish and evaluate the effects on the execution time using 15 web applications.

Our initial results show that we can achieve most of the performance increase with relatively limited resources. We find that 32-128 MB memory, 16 threads, and a speculation depth of 4 is enough to reach most of the performance increase of TLS for the studied web applications.

2. EXPERIMENTAL METHODOLOGY

Our TLS is implemented in the Squirrelfish [14] JavaScript engine which is part of WebKit [7]. We use nested method level speculation and all data conflicts are detected and rollbacks are done when conflicts arise or when we perform commit and we can no longer ensure the sequential semantics.

In Table 2 we selected popular web applications [1] to cover different types of web applications, while making sure that these were being used by a large group of users. We have defined and recorded a set of use-cases for the selected web applications and executed them in WebKit.

To enhance reproducibility, we automatically execute the various use-cases in a controlled fashion [2]. As a result, we can ensure that we spend the same amount of time on various operations. A detailed description of the methodology for performing these experiments is found in [5].

We have extended the Squirrelfish TLS implementation with three parameters that allow us to tune the maximum memory available (in MB), the maximum number of threads, and the maximum depth in nested speculation. When we encounter a JavaScript function suitable for speculation, i.e., that is previously unspeculated we first check the supplied parameters. If the current amount of memory used for speculation, the maximum number of active threads or the maximal speculation depth is below the specified limit, we speculate. If a parameter is above the limit, we do not speculate and continue to execute the function sequentially.

Application	Description
Google	Search engine
Facebook	Social network
YouTube	Online video service
Wikipedia	Online community driven encyclopedia
Blogspot	Blogging social network
MSN	Community service from Microsoft
LinkedIn	Professional social network
Amazon	Online book store
Wordpress	Framework behind blogs
Ebay	Online auction and shopping site
Bing	Search engine from Microsoft
Imdb	Online movie database
Myspace	Social network
BBC	News paper for BBC
Gmail	Online web client from Google

Table 1: Popular web applications used in these experiments

All experiments are conducted on a system running Ubuntu 10.04 that is equipped with dual quad-core processors and 16 GB main memory. The execution time measured is the JavaScript execution time in Squirrelfish, rather than the overall execution time of the whole web application. We execute each case 10 times, and then take the median of the execution time.

3. EXPERIMENTAL RESULTS

In Section 3.1, Section 3.2 and Section 3.3 we have limited the amount of memory available for speculation, the maximum number of threads and the speculation depth. We have measured the execution time relative to the sequential execution time. The horizontal line in the figures is the sequential execution time for comparison. We have also measure the memory usage, the number of speculations and the number of rollbacks.

3.1 Limiting the amount of available memory

The most important results in Figure 1 are: (i) the execution time generally decreases with an increased amount of available memory, and (ii) most of the performance increase is achieved with a relatively small amount of memory (32MB - 128MB).

We see that for all cases except two, the execution time does not decrease beyond 512MB. This indicates that for most of the cases we do not need more than 512MB in order to decrease execution time with Thread-Level Speculation. For *wikipedia* and *bing* there is no change in the behaviour when we increase the allowed amount of memory. Therefore these cases do not need more than 4MB to decrease the execution time.

An interesting case is *amazon* where the execution time is lower than the sequential execution time for 4MB, then the execution time gradually increases up to 64MB, and from that point it gradually decreases until it reaches a point were there is not set any restrictions on the amount of available memory. The execution time is only lower than the sequential execution time for 4MB and when no limitation is set on the amount of available memory. Between 8 and 512MB, the execution time is higher than the sequential one.

This means that the speculation process, while being able potentially to speculate on many JavaScript functions, does not automatically decreases the execution time. By looking at the number of rollbacks, we see that for *amazon* the number is quite high. For the no restriction, we see that the number of speculations is around 2000 higher than when a limitation of 512MB is set. Still the number of rollbacks is the same for both. This suggest that we are able to speculate on a sufficiently large number of function calls, which in turn allows us to have a decrease in the execution time, and makes the execution time lower than the sequential execution time. For *amazon* we see that it is faster (or similar) to the sequential execution time for a limitation of 4MB and no restriction. The first one hardly speculates, and does not have any rollbacks. The last one speculates, but the relationship between rollbacks and speculation makes it possible to make it speculate on a sufficient number of functions making the execution time lower than the sequential execution time.

The relationship between increasing the amount of used memory and the decrease in execution time is however not a continuous one for all of the cases. For instance for *google* and *linkedin*, the execution time is lower for 8MB than for 16MB, for *msn* the execution time is lower for 32MB than for 64MB, and for *bcc* 512MB it is lower than when no restrictions are set.

In Figure 1 we see that there is a clearer correlation between an increase in memory size and the highest number of threads than between an increased memory size and a decreased execution time. There is a limitation to the number of threads that are possible to extract when we increase the memory size. We see in Figure 1 that only 5 of the web applications are able to use more than a maximum of 50 threads regardless of the amount of available memory for speculation.

For some of the web applications there is a relationship between an increased number of threads and a decreased execution time (but this is not true for all). However, the gain in decreased execution time when the number of threads increases often becomes quite small. If we look at the average number of threads, we see that for many of the applications it is sufficient with 100 threads to get the lowest execution time.

In Figure 2 we see the number of speculations and the number of rollbacks when we limit the memory size. There is a clear correlation between an increased memory usage and an increased number of speculations. Unfortunately, we see that an increasing amount of memory, which means more speculations, also means an increased number of rollbacks. However, comparing the number of speculations and the number of rollbacks in more detail, we find that very few of the speculations result in a rollback. For example, *imdb* has at most approximately 5000 speculations, while it only has approximately 150 rollbacks at most. This corresponds to a misspeculation rate below 3%. The results for the other web applications are similar. If we compare the number of

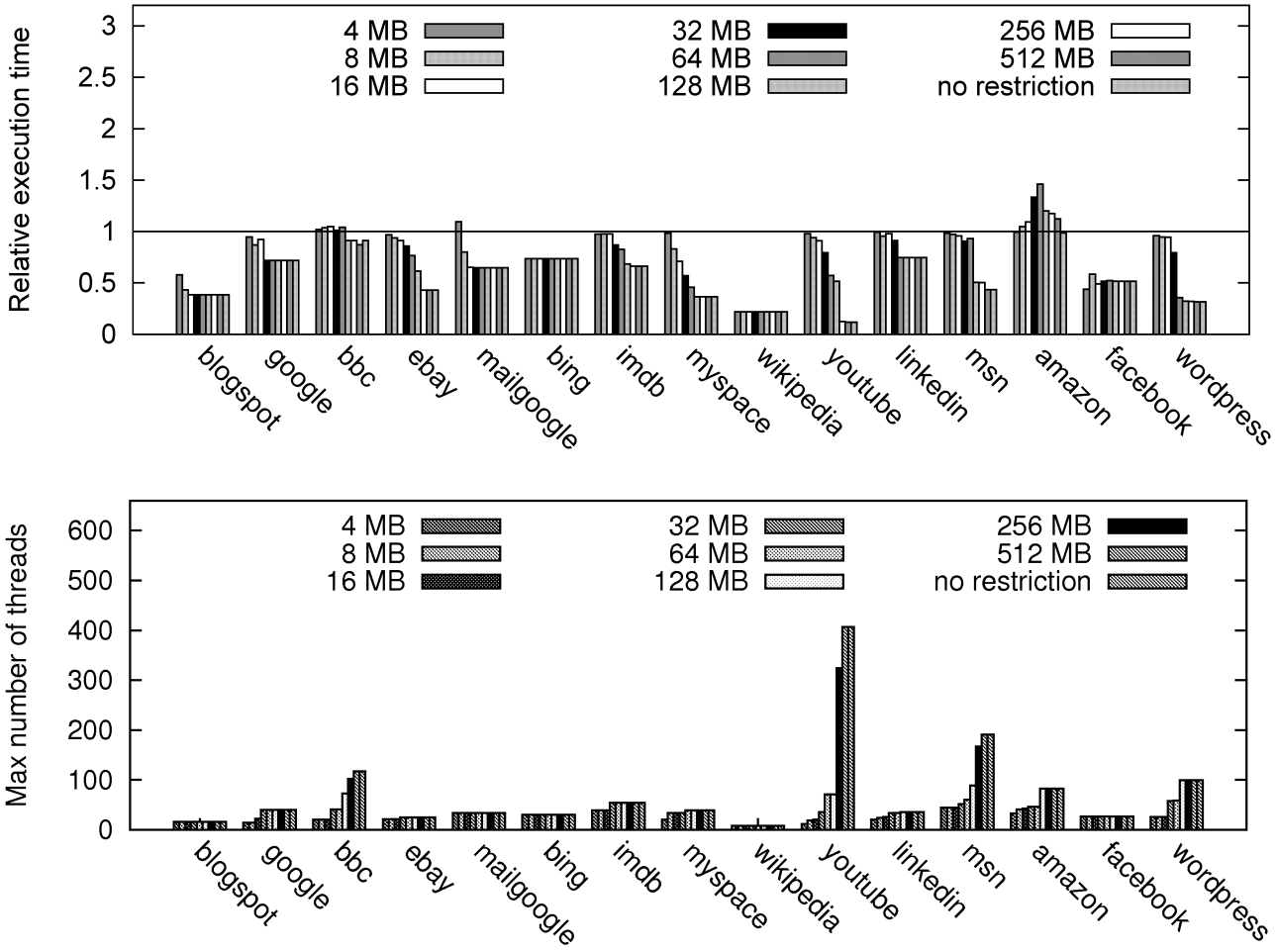

Figure 1: The relative execution time (upper) and the highest number of threads (lower) when we limit the available memory to $4, 8, 16, 32, 64, 128, 256, 512$**MB and with no restriction on the memory usage**

speculations when the available memory increases and the increased number of rollbacks, we see that the number of speculations increases at a higher rate than the number of rollbacks.

In summary, the most striking observation from limiting the amount of memory is that in order to decrease the execution time, it is often sufficient with between 32 and 128MB to get most of the performance improvements. Looking at the overall results from limiting the amount of memory, we observe that in order to have the lowest possible execution time it is important to have a relatively large number of threads running simultaneously, a sufficient number of speculations and a low number of rollbacks. However, it is not necessary to use a very aggressive speculation scheme with unlimited memory resources.

3.2 Limiting the number of available threads

In Figure 3, we present the relative execution time and memory overhead when we limit the number of parallel threads for speculation. The most important observation is that only a small number of threads are necessary in order to achieve most of the performance increase. Our results indicate that 16 threads seem to be enough for the studied web applications. The optimal number of threads in order to decrease the execution time mostly is between 8 and 32, since 13 out of 15 have the lowest execution time with this number of threads. The results in Figure 3 show that only *youtube* is able to take advantage of more than 128 threads. However, the decrease in execution time by increasing the number of threads from 32 to 128 is very small. We have also observed a clear relationship between an increased memory usage and an increased number of threads. This can be understood by that a large number of threads, means a larger number of speculations, which means a larger number of information stored incase of a rollback.

In Figure 4 we present the number of speculations and the number of rollbacks for different maximum number of threads. We see that the number of speculations increases with the number of threads. However, for 12 out of 15 web applications, the number of speculations does not increase when the maximum number of threads is over 16 threads. This

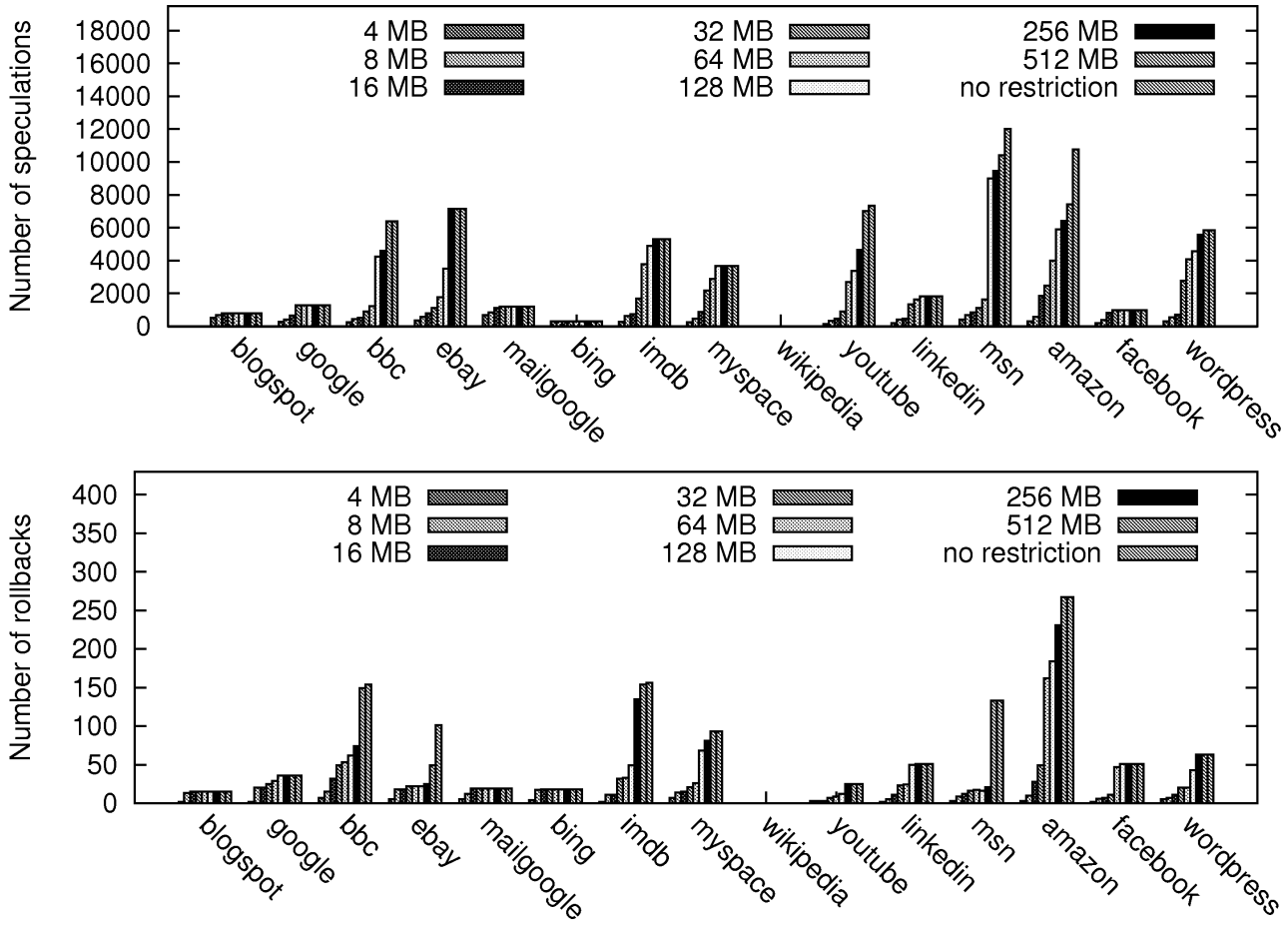

Figure 2: The number of speculations (upper) and the number of rollbacks (lower) when we limit the memory usage to $4, 8, 16, 32, 64, 128, 256, 512$**MB and with no restriction on memory usage**

suggests that we are not able to find a sufficient number of functions to execute concurrently.

From the number of rollbacks we see that there is a significant increase in the number of rollbacks from 2 to 8 threads. However, there is often a decrease in the number of rollbacks as the maximum number of threads increases from 16 up to no limitation on the number of threads. This pattern is quite common, first the number of rollbacks increases, then after the number of rollbacks reaches a maximum, the number of rollbacks gradually decreases as the number of threads increases. If we compare the graphs in Figure 4, we see that there is not a clear correlation between an increased number of speculations and an increased number of rollbacks. This suggests that a larger number of threads does not necessarily mean a larger number of rollbacks. In fact, if we have a certain number of threads it might mean the opposite, an increased number of threads might even reduce the number of rollbacks.

If we restrict the number of threads, we see that for 10 out of the 15 cases, no restriction on the number of threads does not yield the lowest execution time. This means that having a higher number of threads does not necessarily mean a lower execution time. Another observation is that for 11 out of the cases, the highest execution time is with a restriction on two threads. Which suggests that the number of threads must be higher than two in order to take advantage of TLS and decrease execution time.

These observations mean that uncritically increasing the number of threads might not be beneficial to decrease execution time. The optimal number of threads to decrease the execution time is between 8 and 32. A maximum number of threads set to less than 8 could mean that we are not able to create a sufficient number of threads and therefore are unable to decrease the execution time. Further, we have also observed that increasing the maximum number of threads over 32 may even decrease the execution time.

In Figure 7 we have measured the relative increase in maximum number of threads when we tune the limit on the number of threads and we see that for up to 32 threads most cases are able to double the maximum number by tuning the amount of available threads. This suggest that execution wise there is a limitation on increasing the available

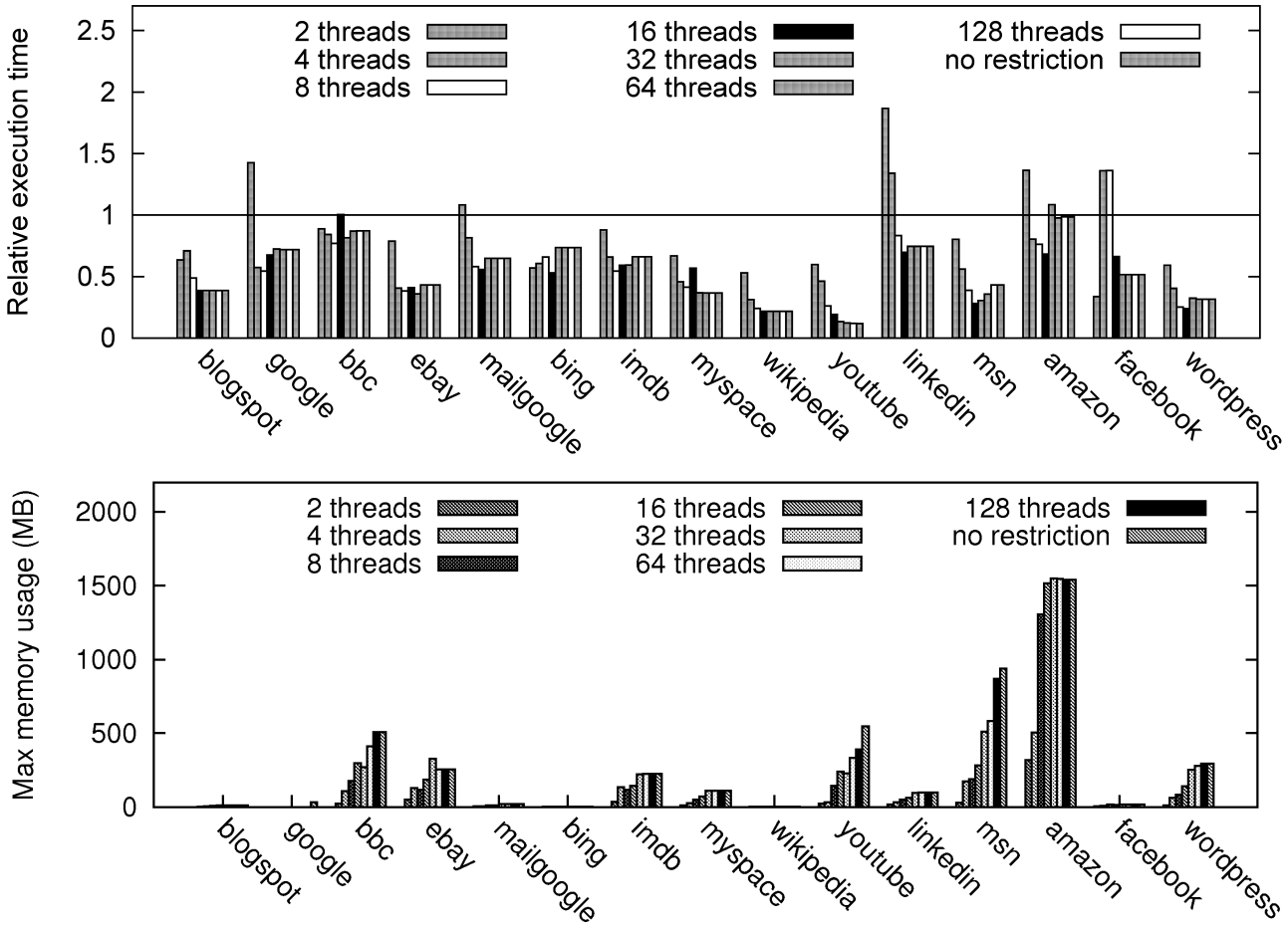

Figure 3: The relative execution time (upper) and the memory usage (lower) when we limit the number of threads to $2, 4, 8, 16, 32, 64, 128$ **and with no restriction on the maximum number of threads.**

threads beyond 32. For *youtube*, we are able to increase it beyond a factor 2 going from 128 to 407 threads. Execution time wise, the increase in the number of threads in this case does not make any real difference.

3.3 Limiting the depth of speculation

From Figure 5 we see that the execution time is the highest at depth 1 (i.e., no nested speculation). This observation indicates that nested speculation is necessary to decrease the execution time. However, the execution time increases compared to the lowest execution time when we speculate too deep, so the largest depth does not mean the lowest execution time. For example, we are often not able to take advantage of very deep nested speculation, and for most of the cases we are not able to gain anything by decreasing the speculation depth below 16. Therefore, there is a penalty to speculating too deep, and the optimal speculation depth in relation to execution time is between 4 and 16.

In Figure 6 we have measured the relative increase in the maximum number of threads running simultaneously when we tune the speculation depth and we see the relationship between the depth and the maximum number of threads that we are able to use. We see that the number of threads increases together with the depth, before it flattens around depth 16. This suggests that there is a limitation to the number of speculations, and thereby the function calls we are able to extract with the nested speculation. However interestingly this limit is not at depth 1, so nested speculation is important in order to find a sufficient number of function calls to speculate on.

In Figure 8 we see that the number of speculations (and to a certain degree the number of rollbacks) decreases as the speculation depth increases. This suggests that the number of rollbacks, and the conflicts does not increase with an increased speculation depth. This means that with an higher depth on speculations, functions that will run as threads will have less chance of a conflict with one another, which in turn means less likeliness for a rollback. This, in addition to finding a larger number of functions to speculate on, is one of the advantages of nested speculation. We also see that there is a clear relationship between an increasing depth and an increased memory usage.

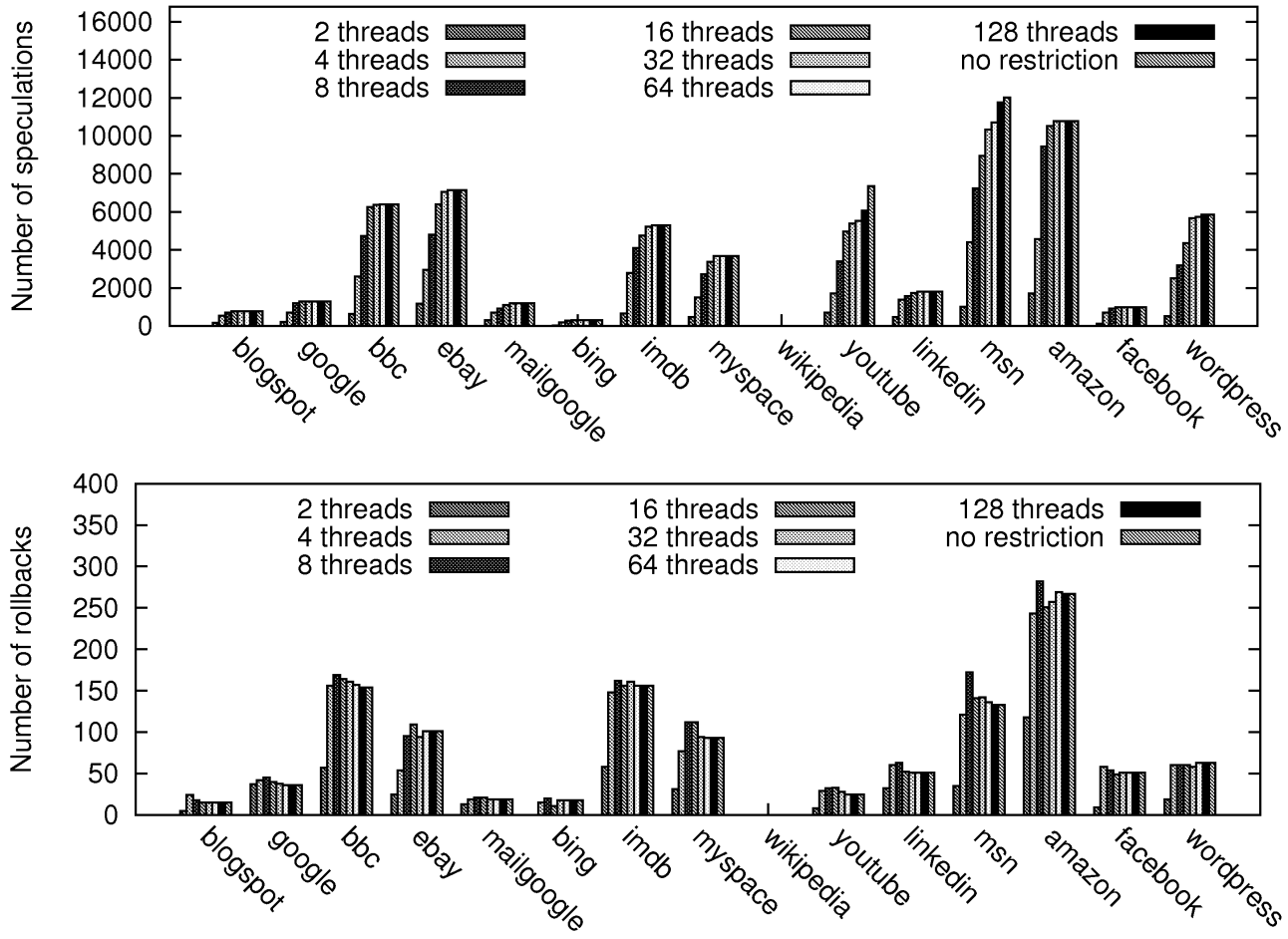

Figure 4: The number of speculations (upper) and the number of rollbacks (lower) when we limit the maximum number of threads to $2, 4, 8, 16, 32, 64$, 128, and with no restrictions on the number of threads.

4. CONCLUSION

In this study, we have evaluated the performance effects when limiting the available memory, the number of threads, and the depth of nested speculations. The results indicate that for the majority of the cases, it is not beneficial to use a too aggressive speculation scheme, since tuning these three parameters can decrease the execution time while decreasing the available memory. We have found that 16 threads, 32MB–128 MB of memory, and a speculation depth of between 4–16 often result in the best performance. Finally, our results show that nested speculation is necessary in order for Thread-Level Speculation to be beneficial.

Our results show that TLS is a suitable technique for increasing the performance of web applications on embedded devices with multicore processors, and where a limited amount of memory is available. The experimental evaluation is performed with eight cores on a dual quad core processor. From the number of speculations and the number of threads running concurrently, there is a clear indication that there is a large potential for an even higher performance with an increased number of cores.

Acknowledgments

This work was partly funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering, (`http://ease.cs.lth.se`).

5. REFERENCES

[1] Alexa. Top 500 sites on the web, 2010. `http://www.alexa.com/topsites`.

[2] J. Brand and J. Balvanz. Automation is a breeze with autoit. In *SIGUCCS '05: Proc. of the 33rd Annual ACM SIGUCCS Conf. on User services*, pages 12–15, New York, NY, USA, 2005. ACM.

[3] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers. A limit study of javascript parallelism. In *2010 IEEE Int'l Symp. on Workload Characterization (IISWC)*, pages 1–10, Dec. 2010.

[4] Google. V8 JavaScript Engine, 2010. `http://code.google.com/p/v8/`.

[5] J. K. Martinsen and H. Grahn. A methodology for evaluating JavaScript execution behavior in interactive web applications. In *Proc. of the 9th ACS/IEEE Int'l Conf. On Computer Systems And Applications*, pages 241–248, December 2011.

[6] J. K. Martinsen, H. Grahn, and A. Isberg. A comparative evaluation of JavaScript execution behavior. In *Proc. of the*

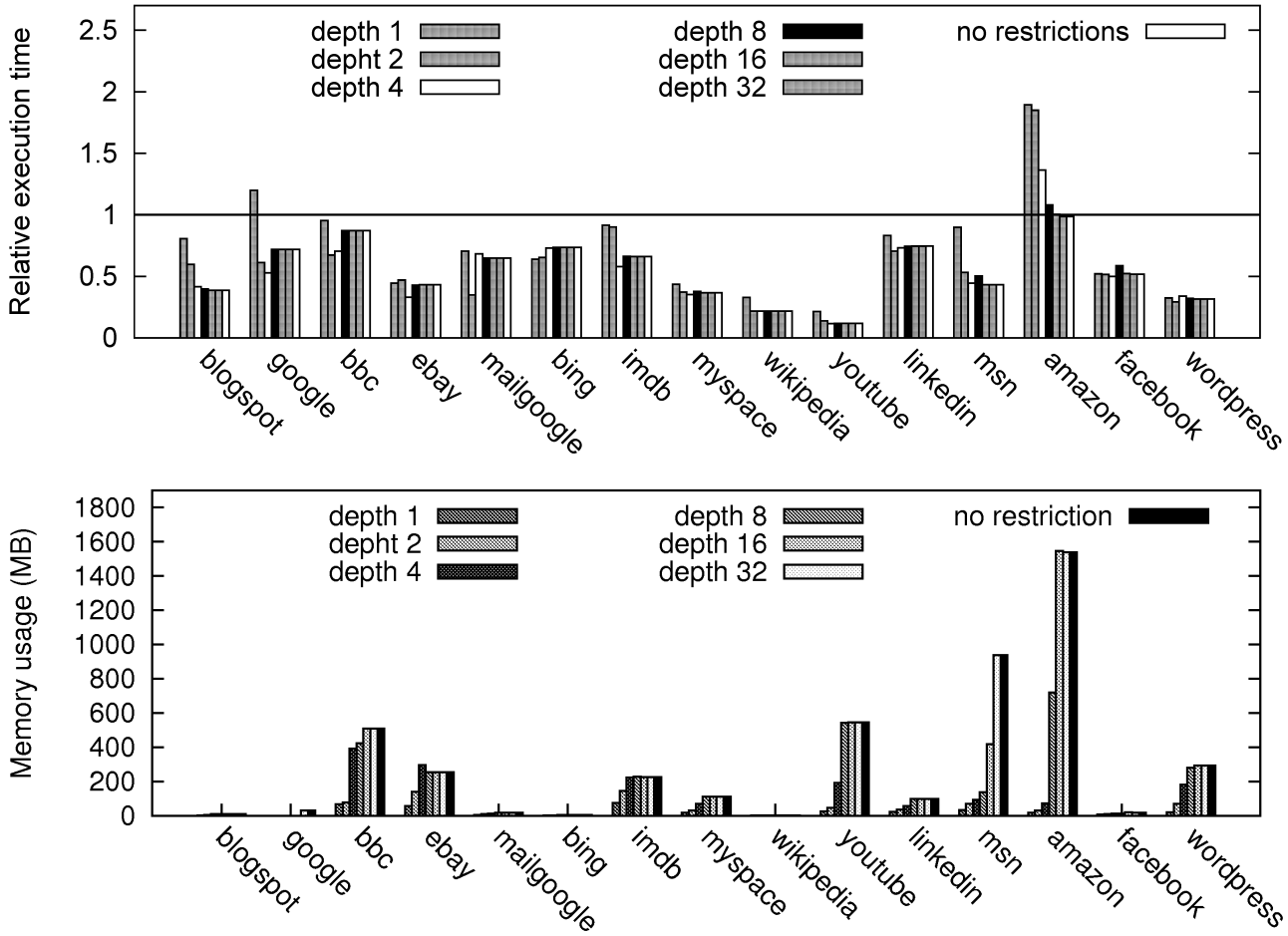

Figure 5: The relative execution (upper) time and memory usage (lower) when we limit the speculation depth to $1, 2, 4, 8, 16, 32$, and with no restriction on the depth.

11th Int'l Conf. on Web Engineering (ICWE 2011), pages 399–402, June 2011.

[7] J. K. Martinsen, H. Grahn, and A. Isberg. The effect of thread-level speculation on a set of well–known web applications. In *Fourth Swedish Workshop on Multi-Core Computing (MCC-11)*, November 2011.

[8] Mozilla. What is SpiderMonkey?, 2010. `http://www.mozilla.org/js/spidermonkey/`.

[9] C. E. Oancea, A. Mycroft, and T. Harris. A lightweight in-place implementation for software thread-level speculation. In *SPAA '09: Proc. of the 21st Symp. on Parallelism in Algorithms and Architectures*, pages 223–232, August 2009.

[10] C. J. F. Pickett and C. Verbrugge. Software thread level speculation for the Java language and virtual machine environment. In *LCPC '05: Proc. of the 18th Int'l Workshop on Languages and Compilers for Parallel Computing*, pages 304–318, October 2005. LNCS 4339.

[11] P. Ratanaworabhan, B. Livshits, and B. G. Zorn. JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. In *WebApps'10: Proc. of the 2010 USENIX Conf. on Web Application Development*, pages 3–3, 2010.

[12] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In *PLDI '10: Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 1–12, 2010.

[13] P. Rundberg and P. Stenström. An all-software thread-level data dependence speculation system for multiprocessors. *Journal of Instruction-Level Parallelism*, pages 1–28, 2001.

[14] WebKit. The WebKit open source project, 2010. `http://www.webkit.org/`.

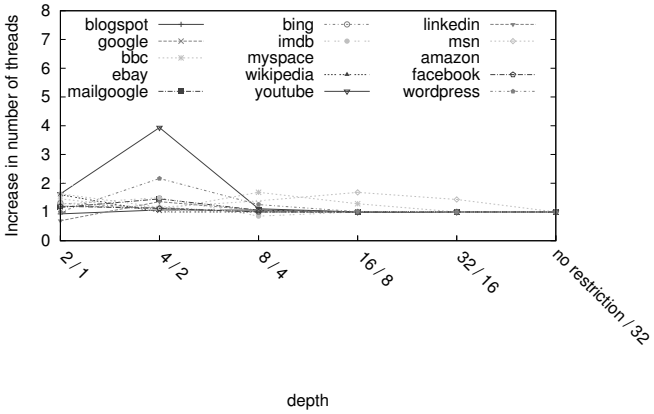

Figure 6: The number of threads we are able to use, and the difference going from one depth to another when we limit the depth to $2, 4, 8, 16, 32$ and no limitation

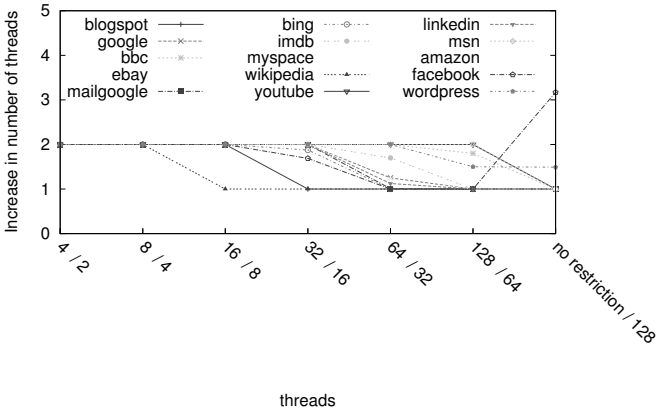

Figure 7: The relative differences going from 2 to 4, 4 to $8, 8$ to $16, 16$ to $32, 32$ to 64, 64 to 128 and 128 to no limitation when we limit the number of threads to $2, 4, 8, 16, 32, 64$, 128 and no limitation

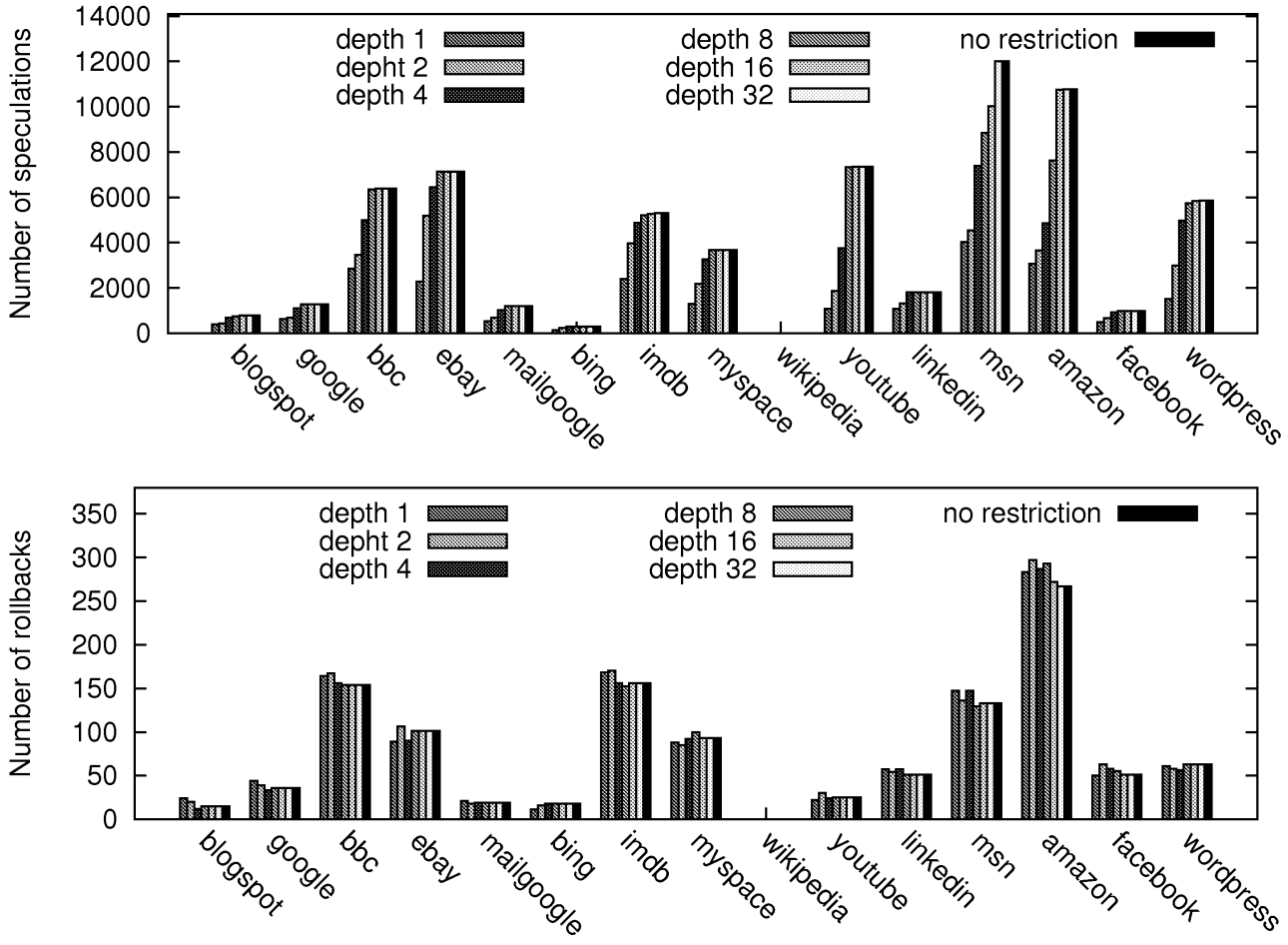

Figure 8: The number of speculations (upper) and the number of rollbacks (lower) when we limit the depth to $2, 4, 8, 16, 32$, and with no restriction on the depth.