Paper Review Form (1000 words maximum)
cs940: Unikernels: Library Operating Systems for the Cloud [1]

January 22, 2018

Paper Summary

3–5 sentences. Briefly summarise the **contributions** *of the paper, i.e., what it adds over the state of the art. Paraphrase and extract the essentials rather than simply copying chunks of text. Be objective; later sections allow for your own opinion.*

Mirage is a *unikernel* software stack targeting the Xen hypervisor that enables the deployment of high performance, efficient, and type-safe OCaml applications. Executing OCaml applications in single-purpose VM appliances, Mirage implements static configuration, compile-time code specialisation, and runtime protection mechanisms. During evaluation, prototype web applications under Mirage achieved comparable or better performance than their equivalent Linux applications, without sacrificing type-safety and security. Drawbacks of requiring extensive OCaml code re-implementation were also discussed, along with the potential of extending Mirage to below-hypervisor level, including on top of the Barrelfish multikernel [2].

Pros and Cons

6 bullets. Succinctly state three positives and three negatives of the paper.

I believe that the paper has the following positive features:

- The paper presented the features and low level implementation details of Mirage in a very comprehensible manner, making effective use of abstract diagrams.
- The authors provided detailed and well-reasoned justifications on trade-offs made between reducing re-implementation complexity, improving performance, and ensuring security.
- The performance improvement provided by Mirage's minimal compilation and runtime environment was well-demonstrated through benchmarking a range of realistic application prototypes implemented in OCaml, with small variations in performance explained through insights on implementation details.

I believe that the paper has the following negative features:

- The OCaml re-implementation complexity can be better measured practically through a case study estimating the cost of re-implementing the full features of a typical Linux application.
- While software written in statically-typed compiled language are easier to re-implement for Mirage, a significant amount of web applications are written in interpreted languages [3]. As a software stack targeting cloud applications, it would be useful to include considerations on the complexity of porting code written in interpreted languages.
- When implementing a prototype DNS server, manual intervention was necessary to improve Mirage's poor performance due to lack of request memorisation. More investigations on whether the same was necessary for other applications to achieve the same level of performance could reveal any hindering language or stack design features.

The Problem/Motivation

1–2 sentences per question. What is the motivation for the work, or the problem being solved? Why is it important? If there is prior art, how was it insufficient? If the problem had not previously been solved, why not?

Modern operating system kernels are large, and contain libraries and features not necessarily required by individual applications. Past efforts to resolve this issue included Drawbridge [4], Singularity [5], and Libra [6]. But between them it was still in lack of a fully self-contained, type-safe solution that also achieves comparable or better performance than conventional Linux applications.

Existing self-contained software stacks [7] [8] serving similar purposes often encounter hardware compatibility issues, therefore a solution targeting a cross-platform system was desired. SPIN [9] was named as a similar idea to Mirage, but the Modula-3 language used by SPIN was quite outdated for re-implementing modern software.

The Solution/Approach

5–10 sentences. What have they done? How does it address the issues set out above? How is it unique and/or innovative (if, indeed, it is)? Give details, again using the paper as the source but again, not just copying text. Instead, focus on paraphrasing/synopsising, and extracting the essential details.

Mirage is designed to operate in a typical hypervisor VM environment, for which it statically define as much of the runtime environment as possible to both reduce cost and to improve performance and type-safety. Unnecessary features are eliminated from the kernel at compile-time, and the code can be optionally sealed with hypervisor support to prevent code injection. Compile-time address space randomisation is also used to improve security.

Mirage is implemented in OCaml, a statically-typed high-performance language suitable for its design goals. It uses Xen-oriented runtime memory management and *Lwt* cooperative

threading. Interactions with device drivers were implemented through shared memory rings, and type-safe I/O handling for networking and storage was used.

The resulting self-contained runtime environment provides a high level of security and stability assurance against code injections and memory corruptions, while also providing excellent cross-platform compatibility by building on top of the Xen hypervisor.

Evaluation

3–4 sentences. How do they evaluate their work? What questions does their evaluation set out to answer? What does their evaluation say about the strengths and weaknesses of their system? What is your opinion of the strengths and weaknesses of the evaluation itself? Give highlights, not a point by point reproduction of the evaluation section(s). In rare cases, systems papers may not have any evaluation, in which case write 'N/A' below.

A staged evaluation process was used to measure and compare the performance of Mirage at different levels of abstraction. Mirage's performance improvements resulting from minimal compilation and maintaining a stable runtime environment were demonstrated through micro-benchmarking against bare-metal and Xen-hosted Linux. It was also shown that Mirage suffered from a minimal overhead in network performance as a result of type-safety. Prototypes of realistic applications were also constructed and benchmarked, with Mirage achieving comparable or better performance than the prototypes' Linux equivalents. Mirage's significantly reduced binary achieved through minimal compilation and dead code elimination was also observed.

Therefore, in the view of the authors, Mirage's performance satisfied their design goals. The practicality of porting existing applications requires some further discussion.

Your Opinion

At least 3 sentences. This is the fun part where you get to judge both the paper and the work it reports! Is the motivation convincing? The problem important? The approach a good or bad idea? Why? Which specific things annoyed you, or you thought were cool, or cool-but-flawed? Justify your opinions! Make an argument which will convince others of your opinion.

In developing Mirage, the authors took an "extreme position" in their design of the software stack, requiring existing codebases to be re-implemented in OCaml for integration into Mirage. Being a minimisation and type-safety-oriented approach that can provide a very high level of runtime security assurance, this approach has its significantly advantages, but also bears noticeable disadvantages in re-implementation complexity that comes as a natural trade-off.

However, in the particular context of cloud applications, it may be difficult for the OCaml-based architecture to gain interest from many interpreted language developers in the industry. The authors did describe a tiered workflow through which a system application written in another language can be gradually ported for running on Mirage. With the development of

engineering tools that can automate some of the porting process, Mirage will stand a greater chance at becoming a commercial reality.

Questions for the Authors

Finally, imagine you're attending a talk about this paper given by one of the authors. Give at least 2 questions that you would like to ask, specific to the paper and the research it reports.

- While OCaml is a statically-typed, high performance language that suits the purpose of Mirage very well, it can contain unknown language-level bugs that impact statically-allocated memory from time to time, such as the *Scanf.fscanf* memory leak a few years ago [10]. What precautions have been or can be taken in Mirage to mitigate the effects of unknown memory management bugs?
- Statically-typed compiled languages do not enjoy a great deal of success in the modern web application market. Do you think the increased complexity in porting dynamically-typed interpreted languages could hinder the success of unikernel software stacks like Mirage?

References

[1] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 461–472.

[2] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: a new os architecture for scalable multicore systems," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 29–44.

[3] V. Prokhorenko, K.-K. R. Choo, and H. Ashman, "Web application protection techniques: A taxonomy," *Journal of Network and Computer Applications*, vol. 60, pp. 95–112, 2016.

[4] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library os from the top down," in *ACM SIGPLAN Notices*, vol. 46, no. 3. ACM, 2011, pp. 291–304.

[5] G. C. Hunt and J. R. Larus, "Singularity: rethinking the software stack," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 2, pp. 37–49, 2007.

[6] G. Ammons, J. Appavoo, M. Butrico, D. Da Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. Van Hensbergen, and R. W. Wisniewski, "Libra: a library operating system for a jvm in a virtualized execution environment," in *VEE*, vol. 7, 2007, pp. 44–54.

[7] D. R. Engler, M. F. Kaashoek *et al.*, *Exokernel: An operating system architecture for application-level resource management*. ACM, 1995, vol. 29, no. 5.

[8] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE journal on selected areas in communications*, vol. 14, no. 7, pp. 1280–1297, 1996.

[9] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers, "Extensibility safety and performance in the spin operating system," in *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5. ACM, 1995, pp. 267–283.

[10] Ç. Bozman, "Yes, ocp-memprof (s)can(f)!" April 2015. [Online]. Available: https://www.ocamlpro.com/2015/04/13/yes-ocp-memprof-scanf/