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

January 21, 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 an OCaml-based *unikernel* software stack targeting the Xen hypervisor which enables the deployment of high performance, efficient, and type-safe cloud applications. By running OCaml applications in single-purpose VM appliances, and implementing static configuration, compile-time code specialisation, and extensive protection mechanisms, prototype web applications could ran with better performance and safer in comparison with their equivalent Linux applications. Drawbacks of the extensive OCaml re-implementation requirements and the potential of extending Mirage to below-hypervisor level were also discussed.

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 features and low level implementation details of Mirage in a very comprehensible manner, making excellent use of abstract diagrams.
- The authors provided detailed and reasonable justifications on trade-offs taken between re-implementation complexity and performance or security improvement measures.
- The performance improvement provided by Mirage's minimal compilation and runtime environment was well-supported through testing a range of real application equivalents implemented in OCaml, with small variations explained through insights into implementation details.

I believe that the paper has the following negative features:

- The OCaml re-implementation complexity can be better measured 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 vast amount of web applications are written in interpreted languages [2]. As a software stack targeting cloud applications, it would be useful to include considerations on the porting complexity from interpreted languages.
- When implementing a prototype DNS server, manual intervention was necessary to improve Mirage's poor performance through request memorisation. More investigations on whether the same was necessary in other applications to achieve the same level of performance could reveal any language or infrastructure features hindering the performance.

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 problem include Drawbridge [3], Singularity [4], and Libra [5]. But there was in lack of a more self-contained, type-safe solution that also achieves better performance than standard Linux applications.

Existing self-contained software stacks [6] [7] serving similar purposes often encounter hardware compatibility issues, therefore a solution targeting a cross-platform system was desired. SPIN [8] was named as a similar idea to Mirage, but the Modula-3 language used by the former is 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 at compile-time from the kernel, and the code can be optionally sealed to prevent code injection. Compile-time address space randomisation is also used to improve security.

Mirage was 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 were used.

The resulting self-contained runtime environment provides a high level of security and stability assurance, while also provides 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. Micro-benchmarking against the same OCaml code running on bare-metal and Xen-based Linux demonstrated Mirage's improved performance through minimal compilation and maintaining a stable runtime environment. It was also shown that Mirage suffers from a minimal overhead in network performance. Prototypes of realistic applications were also constructed, with Mirage achieving comparable or better performance than the prototypes' Linux equivalents. Mirage's significantly reduced binary size through minimal compilation and dead code elimination was also observed.

Therefore, in the views of the author, it was apparent that Mirage satisfied their design goals. The practicalities of porting existing applications require some further discussion as followed in the next section.

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 structuring the software stack, requiring existing codebases to be re-implemented in OCaml for integration into Mirage. This approach has its significant advantages being a fresh, type-safety-oriented approach that can provide a very high level of runtime safety assurance, 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 of the vast amount of interpreted language developers in the field. Although the authors did describe a tiered workflow in which a system application can be adapted for running on Mirage. With the development of engineering tools that can automate some of the 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 language-level bugs that impact memory management from time to time, such as the *Scanf.fscanf* memory leak a few years ago [9]. What precautions have been or can be taken in Mirage to mitigate their effects?
- Statically-typed compiled languages do not find 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 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] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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