CloudProphet: Towards Application Performance Prediction in Cloud

Ang Li, Xuanran Zong, Srikanth Kandula[†], Xiaowei Yang, Ming Zhang[†]
Duke University
Durham, NC
{angl,xrz,xwy}@cs.duke.edu
[†]Microsoft Research
Redmond, WA
{srikanth,mzh}@microsoft.com

ABSTRACT

Choosing the best-performing cloud for one's application is a critical problem for potential cloud customers. We propose CloudProphet, a trace-and-replay tool to predict a legacy application's performance if migrated to a cloud infrastructure. CloudProphet traces the workload of the application when running locally, and replays the same workload in the cloud for prediction. We discuss two key technical challenges in designing CloudProphet, and some preliminary results using a prototype implementation.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: General—*measurement techniques, performance attributes*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Design, Performance, Measurement.

Keywords

Cloud computing, performance, prediction.

1. INTRODUCTION

The public cloud computing market has grown dramatically in the recent years. Many companies, including Amazon, Microsoft, and Rackspace, have all released their own public cloud computing infrastructures, such as Amazon AWS, Microsoft Azure, and Rackspace CloudServers. These infrastructures, albeit offering similar services, can diverge significantly in terms of performance [3]. Hence, if a potential cloud customer is planning to migrate her legacy application into cloud, it is critical to know: *which cloud does my application perform best on?*

The most straightforward way is to actually migrate the application to different clouds, so that the customer can field test her application's performance inside the cloud infrastructures. However, migration is not cheap. For infrastructure providers, the whole software stack of the application needs to be deployed on the cloud VMs, incurring huge configuration effort. The application data also need to be migrated to ensure realistic benchmarking, raising the data security concern.

In this work, we focus on *predicting* an application's performance running on a target cloud infrastructure, prior to any migration effort. Depending on accuracy, the prediction result can directly point the customer to the best-performing provider, or at least limit the scope for actual migration and field testing. There are two common approaches for performance prediction. Standard benchmarks [1, 3] can provide a baseline to compare the performance of different providers. However, as the benchmarks are simple by design, it is challenging to map the complex and multi-tiered cloud applications to the limited set of benchmarks. Modeling is another widely used approach to predict the performance of simple computation and I/O-intensive applications [5]. Yet it remains a challenging task to describe the complex and often distributed cloud applications' workload using concise model characteristics.

In this paper, we propose CloudProphet, a performance prediction tool aiming to provide accurate and application-specific prediction results. CloudProphet takes a trace-and-replay approach [4]. During *tracing*, CloudProphet records the detailed workload information and the internal dependency of a representative application run. During *replaying*, CloudProphet runs an agent in the target cloud platform, which emulates the workload of the traced application run by replaying the recorded workload and dependency. The performance of the agent is then used to predict the application performance after migration.

We choose this approach for several reasons. First, it does not require any *a priori* knowledge of the performance characteristics of the target cloud infrastructure, because the agent uses real workload to test the infrastructure's efficiency. This is particularly suitable for the cloud environment, as the performance of a cloud can change frequently due to interferences and equipment upgrades. Second, the approach is less likely to be limited by the complexity of the application. As long as we can trace the correct dependency, the approach should work with applications with multiple components and rich inter-component communication, which is essential for the practicality of the tool.

In the following, we describe several key design challenges of CloudProphet.

2. CHALLENGES

2.1 Non-deterministic Application Workload

For many multi-threaded applications, such as databases and scientific computation tools [6], the workload on each thread depends on the order of the synchronization events (*e.g.*, locks), which in turn depends on how the threads interleave with each other. Due to the performance (e.g. CPU and I/O speed) difference in cloud, the application threads may interleave differently if migrated, which then leads to different workload. In this case, simply replaying the workload events collected locally can result in poor prediction accuracy.

SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.
ACM 978-1-4503-0797-0/11/08.

We first illustrate the problem using a real example. Figure 1(a) shows part of the request handling code of a file hosting application UDDropBox, and (b) shows the lock and I/O events triggered by two application threads during tracing. Note that thread $T2$ fails to acquire the lock and has to sleep for a while before retrying.

A naive replay mechanism simply replays all events in each thread's trace one after another. This may cause the replayed events to diverge from the real events of the application if migrated. For instance, if the cloud VM has faster disk I/O, $T1$'s `unlock` event may happen earlier than the first `trylock` event of $T2$ (Figure 1(c)). In this case, the first `trylock` of $T2$ would succeed, and the application would directly trigger `open` instead of `usleep`. However, if we replay strictly according to the event trace, $T2$ still needs to replay `usleep` and another `trylock` before `open`. This adds unnecessary overhead to $T2$.

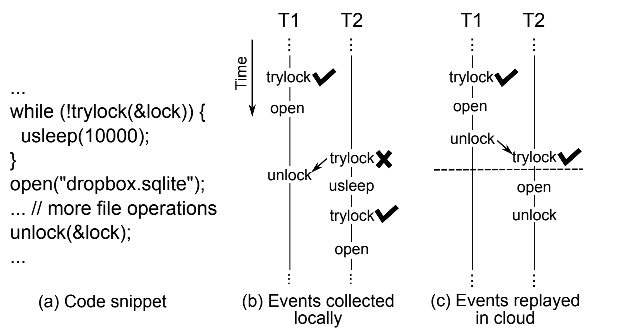
CloudProphet introduces a novel mechanism to address this. During replay, CloudProphet *detects* any synchronization event that occurs out-of-order compared to the order in the recorded trace. An out-of-order synchronization event may cause the future workload events to diverge from the events in the current trace. If such an event is detected, CloudProphet pauses the replay process, and starts a new application run on the local machine to update the workload events after the diverged point. Specifically, the new run is steered by enforcing the new synchronization events order occurred in cloud [2]. After the application run has reached the diverged point, CloudProphet then collects the updated workload events, resumes the replay process, and uses those events for future replaying. The process is repeated if new out-of-order events are detected.

Consider again the previous example. With the new mechanism the cloud replayer will detect that the first `trylock` of $T2$ occurs out-of-order, because the event happens before $T1$'s `unlock` in the trace, while during replay it occurs after. CloudProphet then pauses the replay right after the `trylock`, and tries to update the future events through a new run of the application. During the new run, CloudProphet enforces the order between the `trylock` and the `unlock`, so that the `trylock` always happens after the `unlock`. In this way, the `trylock` is guaranteed to succeed, and therefore the application will immediately trigger `open` and other I/O functions afterwards. Finally, CloudProphet updates the event trace using the new local events that do not contain the extra `usleep` and `trylock`, and resumes the replay.

The mechanism does not make any assumption on the application model, and therefore theoretically works for arbitrary application. On the other hand, one practical limitation is that it requires multiple runs of the application to obtain the right workload to replay. It is our ongoing work to adopt optimizations to reduce the overhead.

2.2 Replay Computation Workload

To faithfully replay the computation (CPU and memory) workload of the original application, we need to trace the exact CPU instructions executed and the memory footprint. However, this can incur significant tracing overhead and increase the replayer complexity. CloudProphet instead adopts a simple linear model to map the local computation workload to the one in cloud. The model scales the CPU time measured locally by a constant factor calibrated by standard CPU benchmarks. The cloud replayer then uses a busy-loop to emulate the scaled workload. The model works reasonably well for most applications we have tested (the error rate is smaller than 30% for most cases), including memory-intensive applications in the SPLASH-2 benchmark [6].



```
...
while (!trylock(&lock)) {
  usleep(10000);
}
open("dropbox.sqlite");
... // more file operations
unlock(&lock);
...
```

Figure 1: (a) The UDDropBox code snippet that accesses the lock-protected database file; (b) The events collected locally; (c) The events replayed in the cloud by CloudProphet.

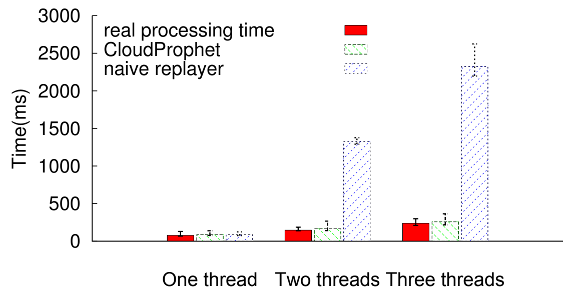


Figure 2: The prediction results of the UDDropBox application.

3. PRELIMINARY RESULTS

Figure 2 shows the prediction results of UDDropBox with 1, 2, and 3 concurrent clients. Each client uploads ten 1MB files back-to-back. We predict the total processing time on an Amazon AWS m1.large instance. In comparison, we also show the real processing time when actually running DropBox on the cloud VM, and the prediction results using the naive replay mechanism. The prediction result of CloudProphet closely matches the real processing time. Moreover, with multiple threads the naive mechanism predicts almost ten-fold of the real processing time, suggesting that the workload collected on our local machine diverges significantly from the real workload in cloud. The result shows our approach is promising in predicting the performance of applications with non-deterministic workload.

4. REFERENCES

[1] Cloudharmony benchmark results. http://cloudharmony.com/benchmarks.

[2] Jong-Deok Choi and Harini Srinivasan. Deterministic replay of Java multithreaded applications. In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools - SPDT '98*, pages 48–59, New York, New York, USA, 1998. ACM Press.

[3] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: Comparing Public Cloud Providers. In *ACM IMC*, 2010.

[4] Michael P. Mesnier, Matthew Wachs, Raja R. Sambasivan, Julio Lopez, James Hendricks, Gregory R. Ganger, and David O'Hallaron. //trace: parallel trace replay with approximate causal events. In *USENIX FAST*, 2007.

[5] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with cart models. In *MASCOTS*, pages 588–595, Washington, DC, USA, 2004.

[6] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ACM ISCA*, 1995.