Brief Announcement: Consistency and Complexity Tradeoffs for Highly-Available Multi-cloud Store

Gregory Chockler[1], Dan Dobre[2], and Alexander Shraer[3]

[1] Royal Holloway, University of London
Gregory.Chockler@rhul.ac.uk
[2] NEC Labs Europe, Heidelberg, Germany
dan.dobre@neclab.eu
[3] Google, Inc.
shralex@google.com

1 Introduction

Cloud storage services are becoming increasingly popular due to their flexible deployment, convenient pay-per-use model, and little (if any) administrative overhead. Today they are being offered by ever growing number of Internet companies, such as Amazon, Google, Microsoft as well as numerous smaller providers, such as Rackspace, Nirvanix and many others.

Although cloud storage providers make tremendous investments into ensuring reliability and security of the service they offer, most of them have suffered from well-publicized outages where the integrity and/or availability of data have been compromised for prolonged periods of time. In addition, even in the absence of outages, the customers can still lose access to their data due to connectivity problems, or unexpected alterations in the service contract (*data lock-in*).

To address these concerns, *multi-cloud* storage systems whereupon data is replicated across multiple cloud storage services have become a hot topic in the systems community. Despite the significant progress in building practical multi-cloud storage systems (see e.g., [1]), as of today, little is known about their fundamental capabilities and limitations. The primary challenge lies in a wide variety of the storage interfaces and consistency semantics offered by different cloud providers to their external users. For example, whereas Amazon S3 supports a simple read/write interface, other storage services also expose a selection of more advanced transactional primitives, such as conditional writes.

In this paper, we outline the results of our recent study [2] that explored the space and time complexity of building reliable multi-cloud storage services.

2 Overview of the Results

Space Bound for Multi-Writer Register Emulations. Our first result establishes a lower bound on the space overhead associated with reliably storing a *single* data item, such as a single key/value pair in a key-value store, supporting basic *put* and *get* operations. For this lower bound we assume underlying storage services exposing put, get, and

Y. Afek (Ed.): DISC 2013, LNCS 8205, pp. 565–567, 2013.

list primitives (such as those supported by Amazon S3), which we model as multi-writer/multi-reader (MWMR) *atomic snapshot* objects. We formalize this setting using the fault-prone shared memory model [4], and prove the following [2]:

Theorem 1. *Let A be a t-tolerant emulation of a wait-free k-writer/1-reader safe register, supporting a set of values V, $|V| > k$, out of a set of $n > t$ wait-free atomic MWMR snapshot objects which can store vectors of length $m > 0$. Then, $k \leq \lfloor (nm - t - 1)/t \rfloor$.*

Our proof constructs a failure and contention-free run α in which all k writers take turns writing into the emulated register each leaving t low-level writes "hanging" on t distinct snapshot objects. We then show that α cannot be extended with another write W as the hung writes may terminate at any time, and in particular, after W returns, erasing all traces of W from the system. Thus, the emulation space overhead is not adaptive to contention. Our result explains the space overheads incurred by recently published practical implementations of reliable multi-cloud stores (e.g., [1]). Their worst-case space complexity is proportional to the total number of writers in the system, which matches our lower bound.

Space-Efficient Emulations Using Conditional Writes. We next turn to emulating reliable registers over storage services supporting transactional update primitives. First, it is well known that a constant number of *read-modify-write* objects is indeed sufficient to reliably emulate multi-writer atomic register [5]. However, the read-modify-write objects employed by the existing implementations are too specialized to be exposed by the commodity cloud storage interfaces. Instead, the cloud storage providers typically expose *general purpose* read-modify-write primitives which are variants of *conditional writes*, and therefore, essentially equivalent to *compare-and-swap (CAS)*.

In [2], we show that there exist reliable *constant* space implementations of *(i)* multi-writer atomic register, which requires the underlying clouds to only support a *single CAS* object per stored value, is *adaptive to point contention*, and tolerates up to a minority of cloud failures and *(ii)* Ranked Register [3] using a single fault-prone *CAS* object. A collection of such Ranked Registers can be used to construct a reliable Ranked Register, from which agreement is built [3]. Our construction thus can be leveraged to implement a multi-cloud state machine replication capable of supporting infinitely many clients with constant space.

Our work opens several avenues for future research. For example, the step complexity of our atomic register implementation is quadratic in point contention. Is this optimal? Interestingly, if this question can be answered in the affirmative, this would imply that there is a time complexity separation between *CAS* and generic read-modify-write primitive, which have been previously thought to be equivalent (e.g., in terms of their power to implement consensus).

Furthermore, our space bound in [2] does not rule out constant space algorithms in which all writers are correct. Since the writer reliability can be enforced in many practical settings, it will be interesting to see whether a constant memory algorithm can be constructed under the assumption of reliable writers, or the space bound can be further strengthened to also apply in this case.

References

1. Basescu, C., Cachin, C., Eyal, I., Haas, R., Sorniotti, A., Vukolic, M., Zachevsky, I.: Robust Data Sharing with Key-Value Stores. In: DSN, pp. 1–12 (2012)
2. Chockler, G., Dobre, D., Shraer, A.: Consistency and Complexity Tradeoffs for Highly-Available Multi-Cloud Store (2013)
3. Chockler, G., Malkhi, D.: Active Disk Paxos with infinitely many processes. Distrib. Comput. 18(1), 73–84 (2005)
4. Jayanti, P., et al.: Fault-tolerant wait-free shared objects. Journal of the ACM 45(3) (1998)
5. Gilbert, S., et al.: Rambo: a robust, reconfigurable atomic memory service for dynamic networks. Distrib. Comput. 23(4), 225–272 (2010)