SyncFree

SyncFree Technology White Paper

Lasp: A Programming Language for Large-Scale Available Systems

Chris Meiklejohn
Université Catholique de Louvain

6 September 2016

Motivation

With the advent of cheap, yet powerful, mobile devices, seemingly overnight, application developers have had to embrace a new form of concurrent programming: namely, distributed programming.

Distributed programming is challenging to get right: multiple clients can be accessing, and attempting to modify, shared resources at the same time and messages between clients can be observed in different orders by different clients. Fundamental to the problem is the notion of partial failure: given I have sent a message to a device asking for a response and have not received it yet, how do I know if that device received the message and is still processing, didn't receive the message at all, or has died?

Additionally, "near-native" experiences have been the gold standard in industry: application developers want their mobile applications to respond to actions immediately, as if remote resources were available locally, to provide a smooth experience to users. To facilitate this "near-native" experience, developers must introduces additional complexity in terms of state: clients will aggressively cache values locally and perform operations on local copies of that state to be synchronized later. What happens when synchronization at a later time determines that these changes are no longer compatible?

Lasp is a new programming language and runtime system for distributed programming that attempts to solve these problems. Lasp attempts to use a principled approach for dealing with highly-available data, data that is replicated and periodically synchronized with convergence rules, for providing an efficient programming environment for building large-scale applications, alleviating the need for users to reason about the uncertainties in distributed programming: an unreliable network where messages between clients may be dropped or re-ordered.

Target Audience

Lasp is targeted at application developers building large-scale applications with replicated, shared state. While the current Lasp prototype implementation is built in Erlang, the techniques that are employed by Lasp can be used across a variety of domains, all of which we're going to explore in the near future: domains, such as, but not limited to, web programming with JavaScript, and mobile programming with Java and Swift.

[1] https://github.com/lasp-lang/lasp

Lasp

Lasp takes a holistic approach at distributed computing by combining two techniques: a **programming language** for authoring of distributed programs, and a **distributed runtime** for the execution of these programs.

Lasp implements a subset of a functional programming language over a limited set of data types chosen for their convergence properties. These data types, known as Conflict-Free Replicated Data Types, are abstract data types extended with functions for conflict resolution for synchronization under concurrent modification. Lasp extends a programming model to these data types, so application developers can create applications as if they are using the normal techniques of functional programming, but the applications themselves are free from concurrency anomalies.

Lasp's distributed runtime system is architected for scale. Leveraging state-of-the-art techniques for reduced state transmission and large-scale membership, applications in Lasp have been demonstrated to scale to single clusters of 500 nodes. Lasp's runtime system is dynamic and configurable at runtime: stable networks can take advantage of optimizations, but Lasp also provides efficient solutions for networks with high churn and failure rates: this allows Lasp to be ideal for stable intra-datacenter focused scenarios, large-latency inter-datacenter focused scenarios, and mobile networks where churn is high and clients are transient.

Lasp is open source and available on GitHub[1]. We provide configurations for running Lasp with Docker and for performing large-scale deployments with the Mesos cluster computing framework.

Positioning

Lasp is a combination of the other techniques from the SyncFree project, combined to form a complete platform for large-scale development of distributed applications. Lasp's goal is to combine the data structures, formal verification and analysis techniques, invariant preservation data structures, and optimized solutions for state dissemination to provide an easy to use, all encompassing solution for application development.

Lasp's system presents a layered approach to the design of the language and runtime solution:

- **Membership Overlay**: Lasp's membership overlay layer is configurable at runtime and supports networks of low churn for use in data centers and networks of high churn for use in large-scale mobile networks [3].

- **Broadcast Overlay**: When the network conditions are ideal, Lasp can optimize dissemination through the use of broadcast primitives build on top of the membership overlay [2, 6].

- **Data Types**: Lasp has support for both pure-op based CRDTs and state-based CRDTs (with $\delta-$CRDTs for efficient incremental state transmission.) [7, 8]

- **Key-Value Store**: For the storage of state, each node runs a Lasp KV store locally.

- **Programming Model**: Lasp provides a functional programming model built on top of a dynamic dataflow execution engine [4, 5].

- **Deployment and Packaging**: Lasp is packaged for Ubuntu, supports deployment with Docker and has prototype integration with the Mesos cloud computing framework for the deployment and operations of large Lasp deployments within the data center [1].

Lasp is a unique offering the space of distributed programming: no system currently exists like it and is the result of collaboration between academia and industry. We've seen many companies offering similar solutions, but most are ad-hoc and don't combine the techniques as Lasp does today.

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *NSDI*, volume 11, pages 22–22, 2011.

[2] J. Leitao, J. Pereira, and L. Rodrigues. Epidemic broadcast trees. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 301–310. IEEE, 2007.

[3] J. Leitao, J. Pereira, and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 419–429. IEEE, 2007.

[4] C. Meiklejohn and P. Van Roy. The implementation and use of a generic dataflow behaviour in erlang. In *Proceedings of the 14th ACM SIGPLAN Workshop on Erlang*, pages 39–45. ACM, 2015.

[5] C. Meiklejohn and P. Van Roy. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*, pages 184–195. ACM, 2015.

[6] C. Meiklejohn and P. Van Roy. Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation. In *Reliable Distributed Systems Workshop (SRDSW), 2015 IEEE 34th Symposium on*, pages 62–67. IEEE, 2015.

[7] P. Sérgio Almeida, A. Shoker, and C. Baquero. Delta State Replicated Data Types. *ArXiv e-prints*, Mar. 2016.

[8] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.