Synchronisation-free programming for large-scale Internet applications

Keywords : Cloud computing; Scalability; Consistency; Programming.

Christopher Meiklejohn, Peter Van Roy

Abstract – Traditional approaches to distributed application design break down for large numbers of devices when there is shared mutable data. The traditional approaches are either not scalable (because they use strong consistency) or extremely difficult to program with (because they use eventual consistency). Synchronisation-free programming provides an alternative that is both scalable and easy to program. This approach is based on the recent discovery of a distributed data structure called a Conflict-free Replicated Data Type (CRDT). Using CRDTs as the foundation, we show how to develop large-scale Internet applications with an easy-to-use programming model, called Lasp, that supports functional reasoning and programming techniques.

Since the advent of Web 2.0, we are witnessing an amazing growth of Internet-based interactive services. Two kinds of service in particular are quite successful at scaling to extreme numbers of clients and amount of data (see Figure 1): "embarrassingly parallel" algorithms (such as MapReduce and SETI@Home) and Content Delivery Networks (such as BitTorrent). It is easy to scale up the former, because by definition they decompose into subproblems that share nothing. It is also relatively easy to scale up content delivery, because the data being distributed is immutable (e.g., movies). In both cases, the only synchronisation needed remains small scale, i.e., between a small number of computing nodes.

But there is an essential, third kind of service that is not covered by these two cases: large-scale applications that use shared mutable data. Simple solutions impose centralisation (inside a data center, see Figure 1), and advanced solutions (using so-called eventual consistency) remain extremely difficult to program and understand. Our research, concretized in the SyncFree project [1], proposes a alternative solution that addresses both issues. Our solution is both much simpler for developers and allows decentralised implementation (close to users, at the "edge").

Consider the simple example of maintaining a shared counter, for example to count advertisement views in an online or mobile application. In a naive implementation, each view remotely increments one shared integer. Serialising updates in this way may work for a low volume advertisement agency, but does not scale up to millions of users. A centralised counter is a serialisation bottleneck, does not support offline mobile devices, and is blocked during a data center failure or disconnection.

Many advanced large-scale applications use another technique, called eventual consistency (EC). EC improves scalabiity by weakening the synchronisation requirements. EC allows concurrent updates to proceed without synchronisation, and propagates and merges them in the background. An EC approach to the advertisement view application might decompose it into partial counters located around the network, would increment each counter locally, and would propagate the partial counts in a peer-to-peer manner. The difficulty here is: how to support high numbers of concurrent, high-volume, continuous flows of count information, without losing an update or counting it multiple times? In the presence of failures, this is extremely difficult to get right.

The SyncFree project uses a new approach to eventual consistency. It avoids the difficulties of the traditional approach while maintaining the advantage of scalability. This approach is called

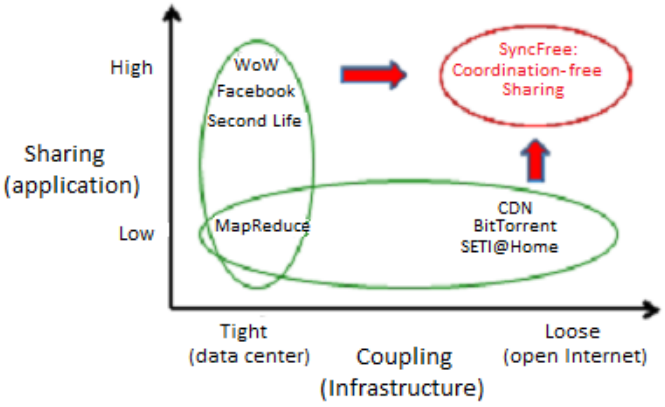

Figure 1: SyncFree project: traditional technologies versus coordination-free sharing.

Conflict-free Replicated Data Types (CRDTs) and it was discovered recently by SyncFree project partners [2]. Using CRDTs makes it possible to do concurrent updates without synchronisation. CRDTs are constructed to satisfy a formal property called Strong Eventual Consistency: if two data nodes see the same updates, then they have equivalent state. Many common data structures, such as registers, counters, sets, graphs, dictionaries, and so on, have been efficiently implemented as CRDTs.

Using CRDTs as the basic data structure, we have developed a programming model for synchronisation-free programming called Lasp [1]. Our initial design provides powerful primitives for composing CRDTs, which lets us write long-lived fault-tolerant scalable distributed applications. Given realistic models of node-to-node communication and node failures, we prove formally that a Lasp program can be considered as a functional program that supports functional reasoning and programming techniques [3]. We have implemented Lasp and have developed several nontrivial applications. We are extending our current prototype into an industrially validated general-purpose language that uses synchronisation as little as possible.

References

[1] SyncFree: Large-Scale Computation Without Synchronisation. EU 7th Framework Project, 2013-2016, https://syncfree.lip6.fr

[2] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. "Conflict-free replicated data types", INRIA Research Report RR-7687, July 2011.

[3] C. Meiklejohn and P. Van Roy, "Lasp: A language for distributed, coordination-free programming", in Principles and Practice of Declarative Programming (PPDP 2015), July 2015.

[1] *Lasp* stands for *Lattice Programming*.