Look Up!

Your Future is in the Cloud

James R. Larus
Microsoft Research
larus@microsoft.com

Categories and Subject Descriptors H.0 [*Information Systems*]: General

General Terms Languages, Management, Performance, Reliability

Keywords Cloud computing; programming languages; distributed computing

1. Talk Abstract

The "Cloud" is a wonderfully expansive phrase used to denote computation and data storage centralized in a large datacenter and elastically accessed across a network. The concept is not new; web sites and business servers have run in datacenters for a long time. These, however, were specialized applications, outside of the mainstream of desktop programs. The past few years has seen enormous change as the mainstream shifts from a single computer to mobile devices and clusters of computers. Three factors are driving this change. 1) Mobile computing, where apps run on a size- and power-constrained device and would be far less interesting without backend systems to augment computation and storage capacity. 2) Big data, which uses clusters of computers to extract valuable information from vast amounts of unstructured data. 3) Inexpensive, elastic computing, pioneered by Amazon Web Services, which enables everyone to rapidly obtain and use many servers.

As a researcher from the language and compiler community, I firmly believe this sea change is at heart a programming problem.[1] Cloud computing is far different from the environment in which most of today's languages and tools were developed, and few programmers have mastered its complexity. New challenges include pervasive parallelism, partial failure, high and variable communication latency, and replication for reliability and throughput.

The Cloud is an intrinsically parallel and distributed world in which computation runs across many processors on many computers. The workload, fortunately, is inherently parallel, as a single server can be shared by hundreds or thousands of simultaneous and independent users. Moreover, the Cloud is built on inexpensive, commodity hardware that fails in perverse ways. Because of scale, failures are common and software must treat failure as normal. At the same time, services in the cloud face stringent requirements to handle large and unpredictable loads while providing consistent response latency and to maintain high availability and reliability. Redundancy and replication are the principal tools to achieve these ends. And, of course, all of these new demands are layered on the familiar challenges of constructing secure, reliable, scalable, elastic, and efficient software! If this is not a programming problem, what is it?

Language designers and implementers have a remarkable opportunity to contribute by providing first-class support and tools to address these new challenges. Languages such as Erlang, C#, Javascript, and Scala have started down this path by supporting asynchronous communication. But, much more can be done. Shared memory and message passing concurrency needs to be integrated into a single, easy-to-use model with an appropriate balance among performance, usability, consistency, and reliability. Richer data models that embrace replication, consistency, partitioning, and persistence would greatly aid software development. And life cycle issues, such as updating continuously running systems and monitoring and controlling large configurations, are appropriate as elements of a language's runtime system.

I will briefly describe Orleans, a software framework developed at Microsoft Research for building reliable, scalable, and elastic cloud applications. Its programming model encourages the use of simple concurrency patterns that are easy to understand and employ correctly. It is based on distributed actor-like components called grains, which are isolated units of state and computation that communicate through asynchronous messages. Within a grain, promises are the mechanism for managing both asynchronous messages and local task-based concurrency. Isolated state and a constrained execution model allow Orleans to persist, migrate, replicate, and reconcile grain state. In addition, Orleans provides lightweight transactions that support a consistent view of state and provide a foundation for automatic error handling and failure recovery. Orleans enables a developer to concentrate on application logic, while the Orleans runtime provides scalability, availability, and reliability. The system is in use by Microsoft, supporting production services.

[1] Yes, it also poses challenges for the networking, database, system management, . . . communities as well.

PLDI'13, June 16–19, 2013, Seattle, WA, USA.
ACM 978-1-4503-2014-6/13/06.