ContextJob: Runtime System for Elastic Cloud Applications

Wei-Chiu Chuang (student), Bo Sang (student), Sunghwan Yoo (student), Charles Killian, Milind Kulkarni*
Department of Computer Science, *School of Electrical and Computer Engineering
Purdue University
{chuangw, bsang, sunghwanyoo, ckillian}@cs.purdue.edu, milind@purdue.edu

1 Introduction

An attractive approach to leveraging the ability of cloud-computing platforms to provide resources on demand is to build *elastic* applications, which can scale up or down based on resource requirements. To ease the development of elastic applications, it is useful for programmers to write applications with simple, inelastic semantics and rely on runtime systems to provide elasticity.

This poster presents ContextJob, a programming model and runtime system for programmers to write applications that conceptually behave as single-node tasks, while the ContextJob runtime allows the application to be safely run in a distributed, elastic manner.

2 ContextJob

ContextJob is developed on top of Mace [2], an event-driven distributed systems toolkit which provides the full set of language, compiler, runtime and distributed services.

The programming model of ContextJob is an extension of InContext model. InContext [4] allows simultaneous events on a physical node to execute in parallel while logically preserving the behavior of atomic event semantics.

In ContextJob, the execution of a single logical node is distributed across multiple physical nodes. A single *head* node serves as the representative of the logical node: all communication with the outside world is intermediated by the head node (*e.g.*, all messages sent to this logical node are routed to the head node, and all messages sent by this logical node are sent from the head node). The head node is also responsible for ordering all events within the logical node by assigning each event a monotonically increasing event number, and requiring events commit in order.

When an event executes, it enters one of the contexts, which is a subset of the logical node state. Since each context is an isolated portion of the entire state, events entering different contexts do not affect each other and can thus run in parallel. However, to preserve the behavior of atomic event semantics, all externally observable effects have to be deferred until events commit.

To facilitate the ease of context programming, we extend the Mace language syntax by allowing programmers to annotate each event transition with the canonical name of context that the transition enters. This information is then translated into a request to acquire the exclusive write lock of the context by the runtime.

The runtime determines how the contexts are mapped to the physical nodes. By associating each event transition with a context, elasticity is achieved by migrating a context from a busy node onto an idle node when demand is high.

ContextJob runtime also includes a job scheduler. The job scheduler is responsible for requesting available resources from parallel batch system such as Condor. It also monitors application status and initiates context migration to idle nodes when needed.

3 Applications

We have built several applications from computational benchmarks to distributed system protocols using the ContextJob system.

MG, for instance, is a widely-used benchmark from NAS Parallel Benchmark suite [1]. MG approximates the solution to a three-dimenisonal discrete Poisson equation using the V-cycle multigrid method.

Tag, a multi-player game server is also coded using context-based approach. The game scene consists of several buildings where rooms are located in each of the buildings. The context'ed Tag models each building as a context, while a room in a building is hierarchically a context below the building context.

One of the best examples to signify the benefit of ContextJob is Paxos [3]. We have implemented an elastic, context-based Paxos. In our design, the elastic Paxos is a lower-level application which provides consensus services to upper-level services. The requests from each upper-level service enter different Paxos contexts, and can therefore execute in parallel.

References

[1] The nas parallel benchmarks 3.3. http://www.nas.nasa.gov/publications/npb.html.

[2] Killian, C. E., Anderson, J. W., Braud, R., Jhala, R., and Vahdat, A. M. Mace: language support for building distributed systems. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation* (New York, NY, USA, 2007), ACM, pp. 179–188.

[3] Pease, M., Shostak, R., and Lamport, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.

[4] Yoo, S., Lee, H., Killian, C., and Kulkarni, M. Incontext: simple parallelism for distributed applications. In *Proceedings of the 20th international symposium on High performance distributed computing* (New York, NY, USA, 2011), HPDC '11, ACM, pp. 97–108.