Scalable, Fast Cloud Computing with Execution Templates

Omid Mashayekhi, Hang Qu, Chinmayee Shah, Philip Levis
Stanford University
(First three authors are currently PhD students.)
{omidm, quhang, chshah}@stanford.edu pal@cs.stanford.edu

Abstract

Today, data analytics frameworks adopt one of two strategies to schedule their computations across workers. In the first strategy, systems such as Spark [3] use a centralized control plane, with a single node that dispatches small computations to worker nodes. Centralization allows a framework to quickly reschedule, respond to faults, and mitigate stragglers. However, the centralized controller can only schedule a few thousand tasks per second and becomes a bottleneck at scale. The second strategy, used by systems such as Naiad [2] and TensorFlow [1], is to use a fully distributed control plane. When a job starts, these systems install data flow graphs on each node, which then independently execute and exchange data. By distributing the control plane and turning it into data flow, these frameworks can execute hundreds of thousands of tasks per second. However, data flow graphs describe a static schedule; even small changes, such as migrating a task between two nodes, requires stopping the job, recompiling the flow graph and reinstalling it on every node.

We present a new point in the design space, an abstraction called *execution templates*. Execution templates schedule at the same per-task granularity as centralized schedulers. They do so while imposing the same minimal control overhead as distributed execution plans. Execution templates leverage the fact that long-running analytics jobs (e.g. machine learning, graph processing) are repetitive, running the same computation many times. Machine learning algorithms, for example, typically iterate until their model reaches an error threshold.

Logically, a framework using execution templates centrally schedules at task granularity. As it generates and schedules tasks, however, the system caches its decisions and state in templates. The next time the job reaches the same part of its program, it instantiates the template rather than resend all of the tasks. Depending on how much system state has changed since the template was installed, a controller can immediately *instantiate* the template, *edit* the template by changing some of its tasks, or *install* a new version of template. Templates are not bound to a static control flow and support data-dependent branches; controllers *patch* system state dynamically at runtime to satisfy the preconditions of installed templates. We call this abstraction a template because it caches some information (e.g., dependencies) but instantiation requires parameters (e.g., task identifiers).

Evaluations of execution templates in Nimbus, a data analytics framework, find that they provide the fine-grained scheduling flexibility of centralized control planes while matching the task throughput of distributed ones. Execution templates support complex, real-world applications, such as a fluid simulation with a triply nested loop and data dependent branches.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM, 2013.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.