Oolong: Programming Asynchronous Distributed Applications with Triggers

Christopher Mitchell cmitchell@cs.nyu.edu
Russell Power power@cs.nyu.edu
Jinyang Li jinyang@cs.nyu.edu

1 Introduction

With the increased popularity of cloud platforms such as EC2 and Azure, application programmers are turning to distributed computation. Existing programming frameworks provide useful abstractions for synchronous computation involving multiple rounds, for which most of a set of input data is examined in each round. For example, MapReduce and Dryad target applications that stream an entire dataset for processing in each round. Piccolo [10] and Pregel [6] rely on global barriers to execute applications such as PageRank and k-means round by round. Asynchronous computation differs from synchronous computation in that the computation does not proceed in lockstep across rounds. Instead, the result of past processing is immediately used to determine the course of current execution. Many problems are solved by efficient asynchronous computation, e.g. single-source shortest path, asynchronous PageRank, web crawling etc. Unfortunately, such computation does not fit into existing programming frameworks that enforce global synchronization.

We present Oolong, a framework designed to address the needs of asynchronous applications. In Oolong, application kernels running on different nodes modify shared state stored in distributed in-memory key-value tables [10]. Since asynchronous applications respond to state changes with further execution, Oolong offers a programming abstraction called the trigger, a section of code that is invoked when data is updated. Triggers can modify updates, insert or update any other state, and schedule additional triggers.

Oolong's design is inspired by database triggers. However, unlike databases, which use triggers in the same transaction as the triggering update to maintain database invariants, Oolong uses triggers to support asynchronous computation, and faces a different set of challenges. To ensure prompt updates while still allowing potentially-blocking user code, Oolong offers two types of triggers that combined provide simple fast updates and complex trigger tasks. Normal triggers are fired and can run short arbitrary code segments when any key-value pair is updated, while long-running triggers asynchronously execute code that performs slow or repeated computations on a per-key basis. To ensure fast recovery from failed nodes, we propose to implement message replication that will allow single failed nodes to be reverted to a previously-saved checkpoint on failure, rather than rolling the whole cluster back [10, 6].

The rest of this WiP explains the design of Oolong in more detail, followed by a brief slice of related research.

2 Design

Oolong is targeted at distributed applications with the following characteristics:

- **Asynchronous:** Many problems are expressed most cleanly as a running queue of pending data to be processed. Data chosen for processing depends on data previously processed, and the execution terminates upon reaching some quiescent state. In these problems, there is no concept of a synchronous round and reduced need for global synchronization.
- **Sparse execution:** Instead of streaming and processing the entire dataset in synchronous rounds, the computation continuously examines subsets of data.
- **In-memory distributed state** The intermediate state must fit in the aggregate memory of all nodes.

An asynchronous application continuously modifies intermediate state, processing and generating updates until convergence. At a high level, Oolong stores such intermediate state in distributed in-memory tables. An application initiates the computation by invoking sections of code called application kernels on one or more workers. In Oolong, application writers also provide sections of code called *triggers*, which are "fired" when an update to a key-value pair occurs, and allow, deny, or modify the update. An active application kernel returns only when all triggers have completed and no new updates are enqueued. In the example of the single-source shortest-path problem, the computation maintains each node's current best path as its intermediate state and kick-starts the processing with a single update on the graph's source vertex. The trigger for shortest-path calculation updates the current best path to a node and generates new updates to the node's neighbors.

Oolong offers two types of triggers. *Short* triggers execute quickly and can be viewed as a generalization of accumulators [10]. Short triggers comprise code to combine old and new values for a key into a new value so that concurrent updates to a key-value pair can occur without synchronization. Additionally, these triggers can enqueue new updates to any table. Sample trigger pseudocode is shown below. *Long-running* triggers

```
def SSSP_Trigger(node_ID, old_dist, new_dist):
  if new_dist < old_dist:
    for target in nodes(node_ID).targets:
      dists.update(target,1+new_dist)
    accept update (new_dist->old_dist)
  else reject update
```

Single-source shortest-path trigger pseudocode

execute code that might otherwise block or delay subsequent triggers. Long triggers run in one or more pertable threads, are associated with a key, and are periodically asynchronously re-triggered. In typical usage, a short trigger may activate a long-running trigger to perform a complex computation or wait for a condition. For example, a web crawler can use short triggers to enqueue links to be examined, and long triggers to perform the crawling. The long trigger could periodically check for the necessary robots.txt file to be retrieved for a site before completing or aborting the actual page fetch.

Oolong offers standard failure recovery based on distributed checkpointing [10, 6]. This strategy can be inefficient because all nodes must roll back to the saved state upon a failure. Oolong therefore also provides finer-grained failure recovery by caching node communication so that one or few failed nodes can be restarted without reverting other nodes. We are considering two designs, one that forwards updates to each worker to a slave worker, and a second in which workers cache outgoing updates. In both cases, a recovered node can be brought up-to-date by replaying incoming updates and discarding duplicated outgoing updates. Such a design poses several questions about performance, overhead, and correctness that we are considering.

3 Applications

We have built several applications to demonstrate the power of Oolong. Oolong has been tested on graph-derived problems such as bipartite matching and single-source shortest-paths. The Python web crawler demonstrated in [10] has been re-written with a simpler Oolong implementation. Finally, we have coded an asynchronous PageRank implementation that is more efficient than standard synchronous iterative PageRank. Qualitatively, our experience suggests that Oolong's mix of traditional synchronous kernels and asynchronous triggers offers a much more general and expressive toolset for application writers. Quantitively, code size between Piccolo and Oolong programs solving the same problem are similar, and the Oolong programs for bipartite matching and shortest-paths computation achieved a $2\times$ to $4\times$ speedup over their Piccolo counterparts by avoiding wasteful reiteration over already-converged state.

4 Related Work

Oolong builds upon the work done for our previous project, Piccolo, which offers a framework on which to build fast, distributed applications centering on the concepts of data partitioning and locality [10]. Piccolo compares favorably on many MapReduce tasks to the Java framework Hadoop, over which it achieves significant speed benefits; it can process a wide range of parallel computation problems that cannot be cleanly or efficiently adapted for MapReduce. Trigger concepts are applied extensively in database systems; McCarthy and Dayal [7] formally define such triggers. The TriggerMan project exploits similarities between many closely-related triggers to reuse trigger code as much as possible, reducing the storage and overhead necessary for many duplicates of the same procedure [3]. Recent systems work has addressed active (event-triggered) procedures in distributed storage [2, 9]. Application-centric distributed systems have explored some of the problems that Oolong addresses with more specific approaches. GraphLab presents a parallel framework for machine learning applications, facilitating simple parallelization of such problems [5]. Network Datalog uses triggers to maintain routing rules in a distributed system, and is tailored for such problems, while Oolong uses triggers as first-class computation citizens [8]. The Linda system leaves framework details such as process coordination to be written by application coders but offers few native features [1]. Mace enables programmers to write network protocols such as Chord as distributed state machines, but is lower-level than optimal for a distributed data-processing framework; Mace allows testing of a failure recovery protocol's details, for example, while Oolong exposes functions to save and load checkpoints [4].

5 References

[1] AHUJA, S., CURRIERO, N., AND GELERNTER, D. Linda and friends. *Computer 19*, 8 (1986), 26–34.

[2] GEAMBASU, R., LEVY, A. A., KOHNO, T., KRISHNAMURTHY, A., AND LEVY, H. M. Comet: An Active Distributed Key-Value Store. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (2010).

[3] HANSON, E. N., CARNES, C., HUANG, L., KONYALA, M., NORONHA, L., PARTHASARATHY, S., PARK, J., AND VERNON, A. Scalable Trigger Processing. In *Proceedings of the 15th International Conference on Data Engineering* (1999).

[4] KILLIAN, C., ANDERSON, J. W., BRAUD, R., JHALA, R., AND VAHDAT, A. Mace: Language support for building distributed systems. In *In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (2007).

[5] LOW, Y., GONZALEZ, J., KYROLA, A., BICKSON, D., GUESTRIN, C., AND HELLERSTEIN, J. M. Graphlab: A new framework for parallel machine learning. *CoRR abs/1006.4990* (2010).

[6] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 International Conference on Management of Data* (2010).

[7] MCCARTHY, D., AND DAYAL, U. The Architecture of an Active Data Base Management System. In *Proceedings of the 1989 ACM Sigmod International Conference on Management of Data* (1989).

[8] NIGAM, V., JA, L., LOO, B. T., AND SCEDROV, A. Maintaining distributed logic programs incrementally. In *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (2011).

[9] PENG, D., AND DABEK, F. Large-Scale Incremental Processing Using Distributed Transactions and Notifications. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (2010).

[10] POWER, R., AND LI, J. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (2010).