Malleable Flow for Time-Bounded Replica Consistency Control

Yuqing Zhu
School of Software
Tsinghua University
Beijing 100084, China
zhu-yq@mails.tsinghua.edu.cn

Jianmin Wang
School of Software
Tsinghua University
Beijing 100084, China
jimwang@tsinghua.edu.cn

Philip S. Yu
Dept. of Computer Science
University of Illinois at Chicago
Chicago IL 60607, USA
psyu@uic.edu

1. INTRODUCTION AND MOTIVATION

Replication is one key technique to guarantee favorable properties of availability, scalability and reliability in large-scale storage systems. According to CAP theorem, such storage systems have to sacrifice replica consistency for availability on network partition. While modern database systems emphasize correctness, completeness and thus consistency, large-scale storage systems challenge modern databases by trading off consistency for availability and guaranteeing only eventual consistency.

However, the weak eventual consistency cannot satisfy all application scenarios. The unknown consistency status on eventual consistency also greatly complicates application development. Developers demand strong consistency occasionally, but strong consistency causes high execution latency and leads to the unavailability under network partition. This cost is highly undesirable. As replica consistency is in a trade-off relation with availability and latency, it is most desirable that the system maximizes consistency within a user-specified latency.

We thus propose the malleable flow (M-flow) approach that can control replica consistency to bound execution time. A system with M-flow supports latency-bounded operations that maximize replica consistency within the given time.

2. M-FLOW: DECOMPOSE & REFORM

The key idea of M-flow is to decompose the replication process into an execution flow of minute steps, and reform a new execution flow by recombining a subset of the steps that are chosen to meet the time requirement and maximize replica consistency status. A better consistency status indicates a shorter latency for a following consistent read; or, a better consistency status means a more recent value returned by a following read within the same latency.

M-flow assumes a symmetric architecture, in which nodes and replicas can be equally accessed. A receiving node without the corresponding data forwards the received request to the closest node holding the replica. Execution results are forwarded back to the receiving nodes and then returned to the requester. The replication process is from when the receiving node receives the request till when the receiving node sends the response.

M-flow first breaks the replication process into six stages, i.e. reception, transmission, coordination, execution, compaction and acquisition. The *reception* stage is when a write is received by a node. The write is transmitted to nodes with the corresponding replica in the *transmission* stage. Writes must go through a *coordination* stage before execution, to guarantee the same write execution sequence for all replicas in the *execution* stage, thus avoiding conflict resolution. The processing of a write can also go through the *compaction* stage that helps to speed up the processing of the *acquisition* stage when the data value is acquired for a read. Stage is the basic unit to guarantee durability and failure tolerance. M-flow then decomposes each stage further into minute steps, each of which requires only limited processing time. The steps and their execution order thus form a directed graph, which consists of six partitions corresponding to the six stages.

M-flow reforms the new execution flow by recombining the right subset of steps, i.e., finding a valid path in the graph. M-flow computes this subset after receiving and before processing the request. The underlying principles are (1) the path either covers or bypasses a stage to guarantee durability and failure tolerance; (2) the step subset is maximized and its execution time is within the given bound; (3) the set of writes processed within a step is maximized; (4) the path for a write covers the reception stage, and that for a read the acquisition stage. To directly improve replica consistency status, the numbers of executed and executable writes are maximized first, thus the maximum number of writes to be executed is computed from stage execution, coordination, transmission, to stage compaction. M-flow borrows the idea from Riemann integral in latency estimation. The small granularity of step enables execution time estimation by simple functions. Besides, recent statistics have greater impacts on the prediction of step execution time. The total latency is the latency summation of all chosen steps.

3. CONCLUSION AND VISION

We have implemented M-flow with a different in-memory storage architecture in the open-source project Cassandra. Experiments over an actual cluster demonstrate that with M-flow, (1) the actual response latency is bounded by the given time; (2) a greater write execution bound leads to a lower execution latency of a following consistent read; and, (3) a greater read latency bound leads to the return of more recently written values.

M-flow is just a first step. One direction for future work is to find a feasible way of integrating this replica consistency variance with the traditional transaction framework. M-flow also opens up a new dimension in providing storage service with regard to response latency and consistency. Exploring this new dimension is yet another interesting future work.

Demo is not planned.

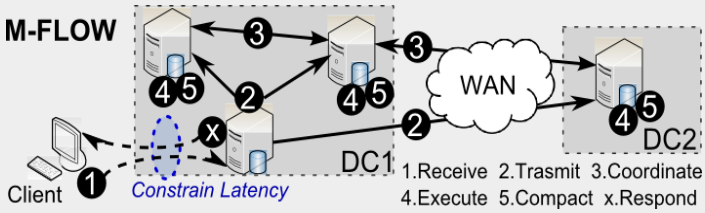
Malleable Flow for Time-Bounded Replica Consistency Control

Yuqing Zhu[1], Jianmin Wang[1], Philip S. Yu[2]
[1]Tsinghua University, School of Software
[2]University of Illinois at Chicago, Dept. of Computer Science

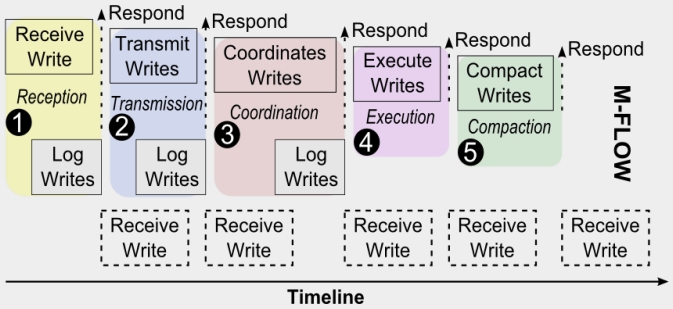
THSS-ISE Lab

Tsinghua University

Big Picture



A system with malleable flow (M-flow) supports latency-bounded operations that maximize replica consistency within the given time. Symmetric architecture is assumed. A receiving node forwards a request to the closest node holding the replica. Execution results are forwarded back to the receiving nodes and then returned to the requester. The replication process, starting from when the request is received till when the response is sent, is decomposed into six stages, i.e. reception, transmission, coordination, execution, compaction and acquisition.



The M-flow replication strategy allows update-anywhere, eager synchronous and lazy asynchronous replication simultaneously. The decomposed replication process enables this flexibility and the control of replica consistency (and latency) by reforming a suitable execution process with carefully selected stages and writes.

* This work is supported by the National Science and Technology Major Special Projects of China under Grant No. 2010ZX01042-002-002-01, and the National Natural Science Foundation of China under Grant No. 61073005.

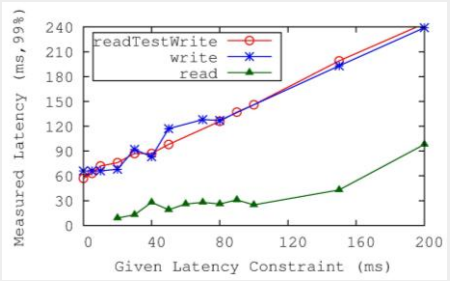
Motivation

- Replication is important for availability, scalability and reliability in large-scale systems.
- Trade-off between replica consistency and availability, latency
- Guaranteeing best replica consistency within a given latency
  - Desirable but not provided

Key Idea

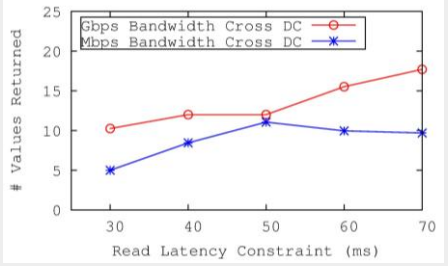
- Decompose the replication process into an execution flow of minute steps
- Reform a new execution flow by recombining a subset of the steps that
  - Meets the time requirement
  - Maximizes replica consistency status
  - Guarantees durability and fault tolerance

Bounded Time & Traded Consistency



Bounded Latencies (the given latency bounds versus the measured 99 percentile latencies):

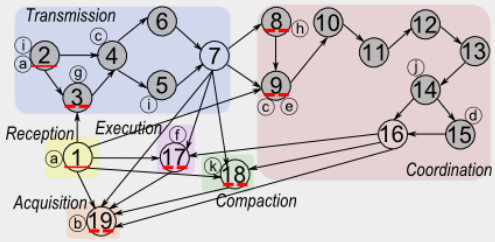
- M-flow can control the general trend on achieving the latency versus consistency trade-offs.



Consistency and latency under different cross-DC bandwidths (the number of returned values vs. the given read latency constraints):

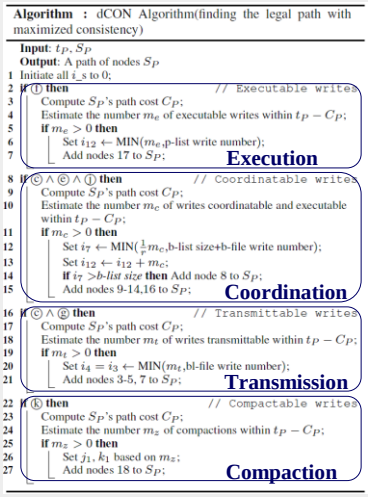
- A larger latency bound for instantaneous reads following writes lead to a larger number of returned values.

The Malleable Flow



The decomposed steps and their execution order form a directed graph, consisting of six partitions corresponding to the six stages.

DECOMPOSE



Principles for reformation:

- The path either covers or bypasses a stage
- The step subset is maximized with execution time bounded
- The set of writes processed within a step is maximized
  - Computed from execution, coordination, transmission, to compaction
  - Increasing executed and executable writes to improve the status of replica consistency
- The path for a write covers Stage reception; that for a read Stage acquisition

REFORM

Implementation over Cassandra

The storage architecture guarantees the durability of writes, and enables the execution flow to stop at the end of any stage.