Failure Detection and Recovery for Doubly Distributed Transactions for Parallel and Distributed Computing

Jay Lofstead* Jai Dayal (student)† Ivo Jimenez (student)‡ Carlos Maltzahn‡
*Sandia National Laboratories gflofst@sandia.gov
†Georgia Tech jdayal3@cc.gatech.edu
‡University of California, Santa Cruz {ivo@cs,carlosm@soe}.ucsc.edu

ABSTRACT

Integrated Application Workflows (IAWs) for processing data prior to storage on persistent media requires scalable mechanisms to ensure operations are completed correctly. Traditionally, processes have used storage systems for the coordination role by placing files or directories as completion signals. With storage systems already a frequent bottleneck in overall system performance and with increasing system sizes, this approach is increasingly inadequate and needs to be replaced with a scalable mechanism that can coordinate the operations of 100,000s of clients and 10,000s servers.

Our previously demonstrated Doubly Distributed Transactions (D^2T) protocol (HPDC 2012 poster and [2]) showed a potential solution, but suffered from scalability limitations and undue server-side requirements. The improved version addresses these limitations and demonstrates scalability with low overhead. (HPDC 2013 poster and [1]). This poster focuses on fault detection and recovery mechanisms implemented and the overheads measured. The performance is similar to the success case with only the addition of appropriate timeouts.

1. INTRODUCTION

IAWs are projected to be essential for scientific computing for extreme scale platforms. These IAWs integrate scientific simulations and data analytics tools into a more tightly connected workflow. Key with this model is the move to eliminate using centralized storage for intermediate data storage between the workflow components. The current standard of using the persistent storage system for staging intermediate results becomes infeasible. Instead, the focus is on developing a new mechanism that replaces the centralized persistent storage mechanism with in compute area data storage and integrated processing.

Existing distributed messaging protocols including Paxos-based implementations like Zookeeper cannot address this situation. D^2T addresses parallel clients while the Paxos-based algorithms all address a single client with multiple server copies. By introducing both parallel clients and distributed or parallel servers, fault detection and recovery are much harder to manage.

The kind of IAW environment we are targeting includes a logically shared data staging area with multiple application clients. Transactions manage data movement from the simulation to the staging area, the data processing in the analysis/visualization routine, and the subsequent data movement to storage. For this evaluation, we focus on the data movement case because it is the most intensive and demonstrates the possibilities at scale.

The poster will include information about the protocol itself to aid viewers. The top level of the hierarchy is the coordinator, the middle layer is sub-coordinators, and the bottom are subordinates (Figure 1). In short, we gather and aggregate up the hierarchy and then broadcast back down.

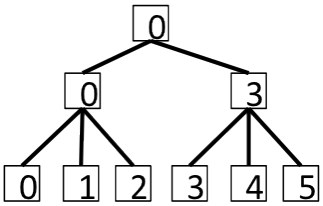

Figure 1: Example Hierarchy

2. FAILURE DETECTION AND RECOVERY

For failure recovery to work, any process must be able to determine what the overall system status was and potentially take over any role. For example, if a sub-coordinator fails, any of its subordinates must be able to step into the role for the system to effectively continue. If the overall coordinator fails, the same holds true. This prompts the requirement that any failure be globally known for the system to be able to recover. Further, the full list of singleton and global transactions must be known by all processes or they cannot step into a role assigned to them. This requirement does incur additional messaging and storage requirements, but there is no other way to guarantee that continued operation or recovery may be possible. This additional messaging has been incorporated in the protocol.

A suitably long timeout must be selected to not cause false positives without introducing undue delays when a failure is detected. Fortunately, the typical and even worst case observed performance is just a fraction of a second. There is an additional challenge introduced by the hierarchical struc-

HPDC2014 June 20, 2014, Vancouver, BC, Canada

ACM 978-1-4503-2505-9/13/11...$15.00.
http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

ture of the new protocol. In the case of a subordinate failure, the sub-coordinator must wait for the timeout interval. The difficulty is that the coordinator is also waiting on the sub-coordinator. That prompts the need for a longer timeout period. The specifics of these periods and how they are handled for each role is detailed below.

While this timeout idea works at a basic level, it is insufficient to handle all of the failure cases. The specifics of how these timeouts are adjusted will be discussed in each case.

2.1 Subordinate Failures

Subordinate failures are the most straightforward because they largely stay localized. In Figure 1, if process 4 fails, the timeout can be localized.

2.2 Sub-Coordinator Failures

When a sub-coordinator fails, both the subordinates that are managed by it and the overall coordinator have to be managed. Consider the failure of process 3 in Figure 1.

In this case, the failure is detected in multiple locations simultaneously. First, all of the subordinates of process 3, processes 4 and 5 in this case, notice that 3 is not responding as expected within the short timeout period. The new sub coordinator tells its peers that process 3 has failed. Just like with the subordinate failure, this notification is sent to all of the subordinates. In this case, because a sub-coordinator failed, the aggregation step never occurred for that sub-tree and the overall coordinator causes the operation overall to fail.

The complication comes in what processes 1 and 2 do because of the timeout. In this scenario, when the gather operation fails, process 0 has already collected the data from 1 and 2, in the role of a sub-coordinator, Processes 1 and 2 are then waiting for the long timeout for the response. The problem is that process 0 is timing out using the long timeout when waiting for process 3. The solution is discussed as part of the poster.

2.3 Coordinator Failures

When the overall coordinator fails, only the sub-coordinators need to worry. However, it is not quite that straightforward. In Figure 1, process 0 represents the overall coordinator.

In this case, the subordinates have all sent out their messages to their sub-coordinators and are waiting for the response back. In this case though, processes 1 and 2 treat the coordinator as a sub-coordinator as well with a short timeout. The problem is that the sub-coordinators also are waiting with a short timeout on the coordinator to accept their aggregated messages. In this case they need to abort their wait and tell their subordinates to abort waiting as well. If they continue to wait, there will not be any possible downward message broadcast possible other than to indicate the failure. In this case, the wait can be short-circuited avoiding any additional wait times. The messaging about the coordinator rank failure triggers this short circuit.

3. EVALUATION

Tests are performed on Cielo, the Cray machine at LANL with tests at up to 65536 clients.

To test the failure recovery process, we inserted code for a single process to skip the rest of the test harness to represent a failure. Otherwise, the tests are identical to those performed for the general evaluation.

The results presented here represent the worst case times for all of the tests. We chose this metric because the impact of the worst case is critical for determining if fault detection and recovery is too expensive. The short timeout value is set to 5 seconds while long is 10 seconds. Results are shown in Figures 2, 3, and 4.

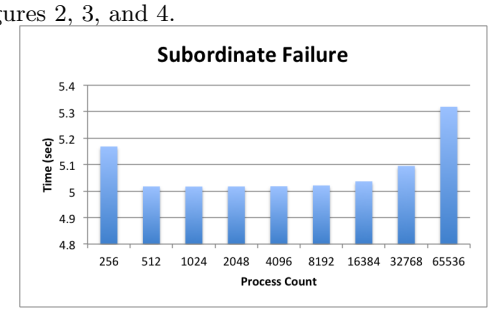

Figure 2: Worse Case Subordinate Failure Overhead

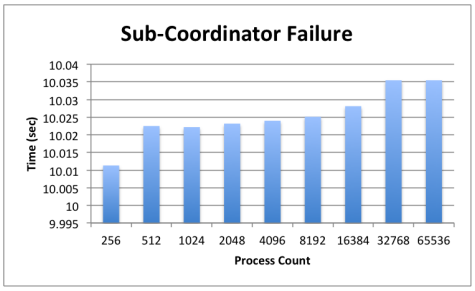

Figure 3: Worse Case Sub-Coord. Failure Overhead

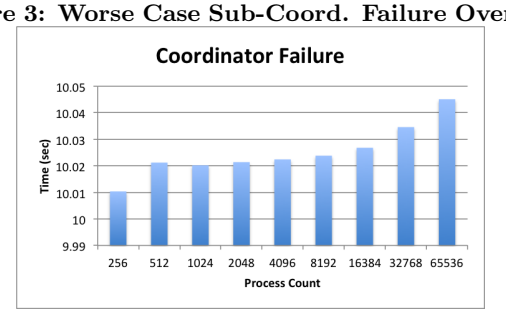

Figure 4: Worst Case Coordinator Failure Overhead

4. ACKNOWLEDGEMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

5. REFERENCES

[1] J. Lofstead, J. Dayal, I. Jimenez, and C. Maltzahn. Efficient transactions for parallel data movement. In *The Petascale Data Storage Workshop at Supercomputing*, Denver, CO, November 2013.

[2] J. Lofstead, J. Dayal, K. Schwan, and R. Oldfield. D2t: Doubly distributed transactions for high performance and distributed computing. In *IEEE Cluster Conference*, Beijing, China, September 2012.