Collective I/O using RAM Area Network

Sergio Servantez
Computer Science Department
Illinois Institute of Technology
Chicago, Illinois 60616
sservantez@hawk.iit.edu
Research Advisors: Dr. Rick Zamora, Dr. François Tessier, Dr. Zhiling Lan

Background

I. Motivation

The objective of this research was to explore the use of remote memory in performing collective I/O operations. The motivation for pursuing this project was driven by two factors. First, there has been a steady growth in the research of disaggregated memory as a means of expanding the conventional memory hierarchy to include a remote level. Research in this area has shown that page-swapping to remote memory typically outperforms a traditional architecture where pages are swapped to disk [1], and that a disaggregated architecture can provide a significant performance-per-dollar improvement [2]. Second, similar research has explored the use of a SSD-based burst buffer as a persistent storage layer where data is staged before asynchronously being transferring to a parallel file system in the background [3]. This new hierarchical storage system outperformed writing directly to the file system in every experiment. Similarly, this project sought to improve I/O performance by using a RAM area network as an aggregation layer for performing two-phase collective I/O operations.

II. Collective Operations

Collective operations improve I/O performance by merging the requests of ranks in a parallel application (see Fig. 1). Typically, these operations are performed in two phases. In the first phase, a subset of ranks are chosen to serve as aggregators where non-sequential data from each rank is organized into contiguous portions of the file. In the second phase, these contiguous portions are written to the file system. The data is moved over several iterations called rounds. In each round, specific chunks of the file are organized into the aggregators. Ranks must remain in sync to ensure the correct data is written to the aggregator at each round. This synchronization among the ranks incurs a cost which represents a significant portion of the overall execution time of the transfer. The crux of this research investigates the use of remote memory to minimize this synchronization cost and by extension improve overall I/O performance.

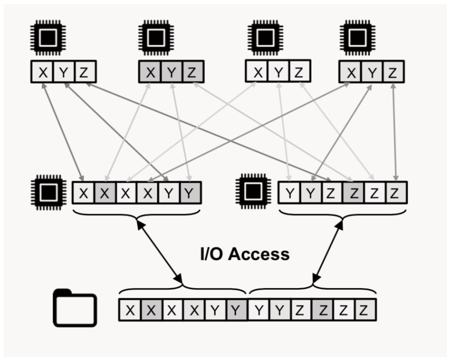

Fig. 1: Two-phase collective I/O write operations

Design And Implementation

I. Design Overview

The initial objective of the research was to create a simple working implementation of the two-phase collective I/O operations using a RAM area network as an aggregation layer. The RAM area network for this project consisted of three Kove XPD devices connected to Argonne's Cooley cluster via an Infiniband network. The initial design was to organize the file into contiguous segments on the remote memory then write those segments directly to the parallel file system. However, we soon discovered that data on the remote memory could not be written directly to the file system. Data would first need to be pulled back to a compute node before it could be sent to the file system. This led to the design of the three-phase collective I/O operation. In the first phase of this design, noncontiguous data from each rank is organized into contiguous segments on the remote memory. In the second phase, the contiguous segments are read back from the remote memory to a subset of selected ranks called aggregators. Last, the aggregators write the contiguous segments to the file system.

II. Synchronous Three-Phase Implementation

At this point in the research, the focus was on creating a proof of concept. There would clearly be an increase in overhead associated with adding an additional phase, but optimization was not the goal at this point. The emphasis was on simplicity so the initial implementation was designed to perform all phases synchronously with no overlap between the phases. Fig. 2 shows experimental data collected from the synchronous implementation of the three-phase collective write operation. In each experiment, the collective write is executed on three nodes with 12 processes per node. 512 MB is being transferred per rank (18 GB total) with a 4-to-1 aggregator ratio.

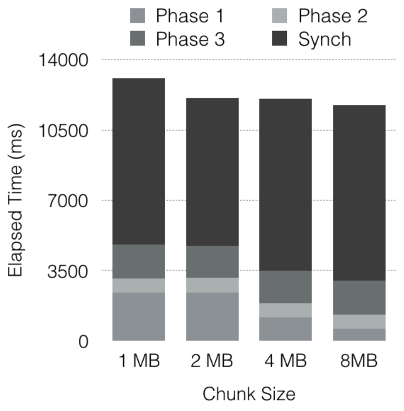

Fig. 2: Performance with varying chunk size

This figure demonstrates how performance decreases as the size of the contiguous segments on each rank decreases. This decline in performance is to be expected since a decrease in segment size translates into higher write frequency to organize the rank's data onto remote memory. Fig. 2 also shows that the preponderance of total transfer time is being spent synchronizing the ranks between rounds. With a working implementation now completed, the focus was shifted toward performance optimization. The traditional two-phase collective I/O algorithm was designed for a traditional computer architecture. The next step in the research was to design a new algorithm which could take advantage of the capabilities and strengths of a disaggregated architecture. Particular consideration was given to minimizing the synchronization cost among the ranks between rounds.

III. Algorithm for Disaggregated Architecture

Expanding the conventional memory hierarchy by adding a remote layer provides compute nodes with a seemingly infinite memory space. This algorithm was designed to take full advantage of this capacity. The algorithm begins by having each rank dump its local data to remote memory. Unlike the traditional two-phase algorithm, the data is not organized into contiguous segments at this first phase. Each rank merely copies its data to remote memory. In the second phase, aggregators are selected and tasked with reading the data back from remote memory. Here is where the contiguous segments are organized. Each aggregator knows what data it needs for any given round and where to find each chunk of data within the remote memory. Once an aggregator has collected all the chunks of a contiguous segment from remote memory, that segment is written to the file system which is the third phase.

Algorithm 1: Collective Write using RAN

```
Function threePhaseWrite($file, data, count, type, status$)
    $volume \leftarrow getVolume()$;
    $rounds \leftarrow calculateRounds()$;
    $burstSize \leftarrow getBurstSize()$;
    $chunkSize \leftarrow getChunkSize()$;
    $asyncWriteToRAN(volume, data, burstSize)$;

    forall $round \in rounds$ do
        if $write\ is\ complete$ then
            $asyncWriteToRAN(volume, data, burstSize)$;
        $wait(round\ data)$;

        if $I\ am\ an\ aggregator$ then
            forall $chunk \in chunks$ do
                $asyncReadFromRAN(volume, chunk, chunkSize)$;
            if $read\ is\ complete$ then
                fileWrite(file, offset, block, status);
```

Fig. 3: New algorithm for collective write on disaggregated architecture

Note that the key advantage to this algorithm is that synchronization cost is all but eliminated. The ranks must coordinate with each other once to signal that all local data has been transferred to remote memory. However after this one and only synchronization barrier, each aggregator is free to collect its needed chunks of data without regard to the other aggregators. This means ranks no longer need to be on the same round for the operation to execute properly and thus there is no need to synchronize between rounds. Moreover, the algorithm calls for all I/O operations to be executed asynchronously. This means the phases will now overlap providing even better performance.

PERFORMANCE EVALUATION

I. Performance evaluation using RAN benchmark

The above algorithm was implemented and tested. Fig. 4 shows experimental data collected from the asynchronous implementation of the three-phase collective write operation. The figure demonstrates the effect on performance when the number of ranks is varied while being executed on three nodes. Each rank is transferring 512 MB with a 4-to-1 aggregator ratio. Fig. 4 shows that performance diminishes significantly once we pass the threshold of four processes per node. We found this to be true regardless of the number of nodes used. While the six and twelve rank experiments fall within the expected performance range, we observe a drastic spike in transfer time once twelve ranks is surpassed. While more ranks would inevitably translate into more network contention, this alone does not explain the performance we are observing. For if the network were truly saturated, we would expect to see proportionate increases in transfer time at every six rank interval. Not only is this not the case, we can

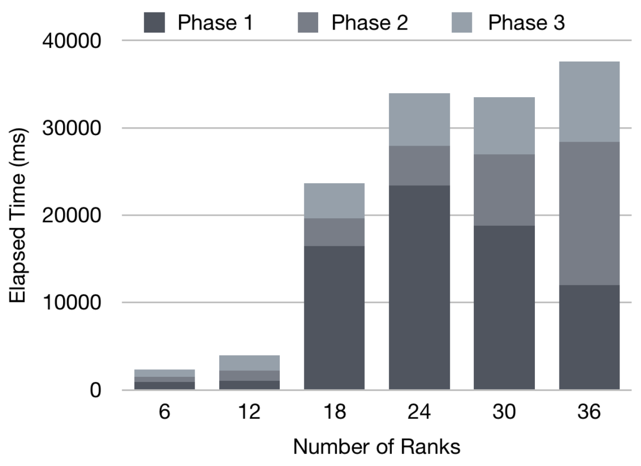

Fig. 4: Performance with varying number of ranks

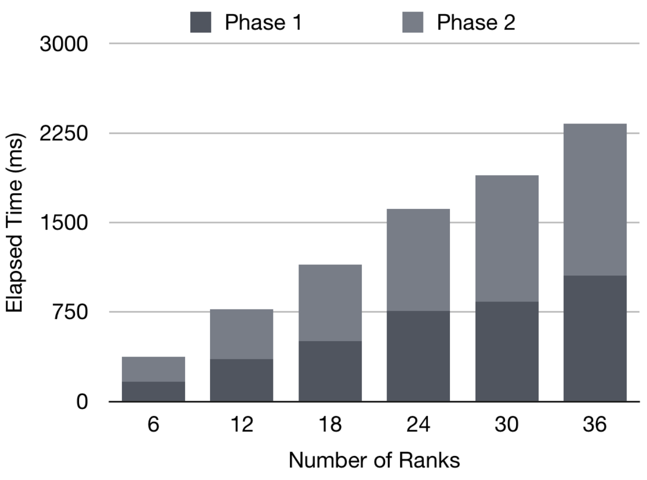

Fig. 5: Benchmark performance with varying number of ranks

observe that the transfer time is relatively constant between the 24 and 30 rank experiments. Output from all experiments was also validated and proved to be valid. So we know the transfer process is still executing correctly. At this point, we can offer no explanation as to why this drastic increase is occurring.

To better understand the expected performance as the application scales, we designed and developed a RAN benchmark. Fig. 5 shows experimental data collected from the RAN benchmark while being executed on three nodes. Each rank is simply writing 512 MB to remote memory then reading that same data back. Phase 3 is not represented in this figure since the benchmark is only testing transfer times to and from remote memory. Fig. 5 shows a near perfect linear increase as the number of ranks increase. For the six and twelve rank experiments, we can observe similar transfer times between the benchmark and asynchronous implementation. However, the benchmark does not demonstrate the same drastic spike to transfer time once we pass twelve ranks.

Conclusion

While this research area still requires further exploration, this project has provided a solid foundation for optimizing collective I/O operations using a RAM area network. As disaggregated architectures become more prevalent, the need for such optimized operations will continue to grow. This research has shown it is possible to perform collective I/O operations with minimal coordination among the ranks. This is particularly notable because synchronization costs typically represents a significant portion of the total transfer time in traditional architectures.

References

[1] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In ISCA '09: Proceedings of the 36th annual International Symposium on Computer Architecture, pages 267–278, New York, NY, USA, 2009. ACM.

[2] K. Lim, Y. Turner, J. Renato Santos, A. AuYoung, J. Chang, P. Ranganathan, T. F. Wenisch, System-level implications of disaggregated memory, Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture, p.1-12, February 25-29, 2012.

[3] B. Dong, S. Byna, K. Wu, Prabhat, H. Johansen, J. N. Johnson, N. Keen. Data Elevator: Low-Contention Data Movement in Hierarchical Storage System. 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), 152-161.