Optimizing Data Locality between the Swift Parallel Programming System and the FusionFS Distributed File System

Mermer Dupree
University of Illinois at Chicago
mdupre2@uic.edu

Justin Wozniak
University of Chicago
wozniak@mcs.anl.gov

Mike Wilde
University of Chicago
wilde@mcs.anl.gov

Ioan Raicu
Illinois Institute of Technology
iraicu@cs.iit.edu

ABSTRACT

Many of the high-performance computing (HPC) systems use a centralized storage system that is separate from the compute system. This approach is not going to be scalable as we seek to achieve exa-scale performance[6]. Distributed file systems can provide the scalability needed for exa-scale computing. FusionFS is a file system designed for HPC systems that achieves scalability in part by removing bottlenecks found in metadata management. Swift/T is a high level, implicitly parallel scripting language for HPC systems. Swift/T provides automated parallelism and load balancing on a massive scale. Additional optimizations can be achieved by utilizing the features FusionFS and Swift/T to take advantage of locality. In this paper, we will look at using Swift/T's language features to optimize locality in FusionFS.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management—*Distributed file systems*

Keywords

Distributed File System, Swift/T,FusionFS, Locality

1. INTRODUCTION

As scientific instruments advance, it becomes increasingly important to be able to process extremely large data sets. At the Advanced Photon Source (APS) at Argonne National Laboratory, X-ray scattering science experiments can produce around 15 TB a week and processing can produce double that data [5]. The increasing demands of processing such massive amounts of data has sparked interest in Big Data research.

```
int X = 1000, Y = 1000;
int A[][];
int B[];
foreach x in [0:X-1] {
foreach y in [0:Y-1] {
  if (check(x, y)) {
   A[x][y] = g(f(x),f(y));
  } else {
   A[x][y] = 0;
  }}
B[x] = sum(A[x]);
}
```

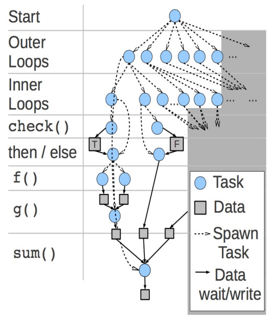

Figure 1: An example of Swift's dataflow[4]

Swift/T is a distributed workflow system that has a high-level scripting language, called Swift, that is simple and intuitive to the user, yet capable of automating the complexities involved in processing massive amounts of data in parallel.

Optimizing Swift/T for large data processing on a distributed file system will involve Swift/T interacting with a distributed file system. In this paper, we will explore the option of using FusionFS and investigate how its unique capabilities make it ideal for optimizing locality and eliminate storage bottlenecks found in other distributed file systems.

2. DESCRIPTION

2.1 Swift/T

Swift/T is a programming model and runtime engine, focused on many-task computing, developed at Argonne National Laboratory. Swift/T's innovations include an easy-to-use high-level dataflow language and a distributed dataflow engine capable of balancing tasks over a large number of nodes[4]. Tests have shown that Swift/T can efficiently scale to 120K compute nodes[4].

Swift/T's massive parallelism is achieved with the Swift scripting language, the STC compiler and the Turbine runtime engine.

The Swift scripting language provides a simple syntax for users. The applications the user runs with Swift/T are

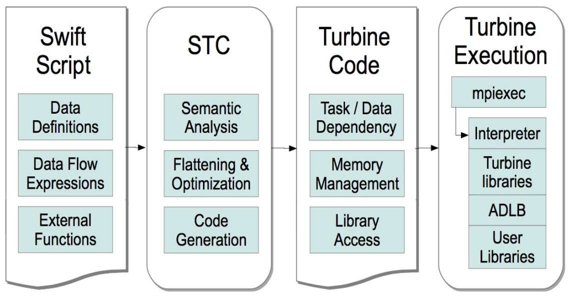

Figure 2: Swift/T's flow chart[1]

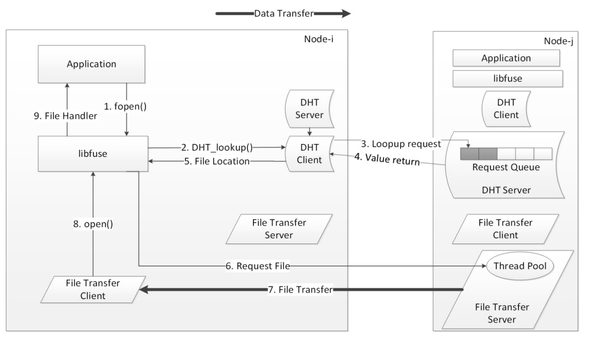

Figure 3: The file open in FusionFS[6]

mapped as "leaf tasks" which can be called in a Swift script like a function. This provides a natural way for users to write their scripts.

Swift/T uses advanced dataflow algorithms to protect data integrity while achieving massive parallelism and efficiency. First the STC compiler translates Swift scripts into Turbine code. STC performs optimizations and divides work into tasks. Turbine runtime engine controls dataflow and distributes tasks to workers. Figure 1 shows the translation of swift scripts into tasks with data dependencies. Tasks subscribe to the variables that make up the taskâĂŹs input. Once a variable is given a value, it is frozen and cannot be changed. When all the variables a task has subscribed to have been frozen, the task is free to be evaluated and is put in a queue. Turbine engines use ADLB to assign tasks to worker nodes and to achieve load balancing. Figure 2 shows a flow chart of what we described.

2.2 FusionFS

FusionFS is a distributed, user-level file system designed for HPC systems. FusionFs is optimized for applications that are write and metadata intensive. We will show that these attributes make it ideal for optimizing locality using Swift/T.

All writes are local in FusionFS. When a file is created by a node in the Fusion directory, the file stays on that node until it is opened by another node. When a file is then opened by another node, the file transferred to that node and the metadata is then updated to reflect the files new location. This allows FusionFS to quickly create and write files.

In FusionFS, both files and metadata are distributed across the nodes. FusionFS decouples the data from the metadata such that the metadata that is stored on a node is not necessarily the metadata associated with the files stored on the node [6]. Metadata is distributed among the nodes using a distributed hash table. A file path is used as a key and is hashed to a node, which contains the metadata for that file. FusionFS uses ZHT (Zero-Hop Hash Table) as its distributed hash table. ZHT provides many features that are important to HPC systems such as being light-weight and fault tolerant, and providing consistent hashing, scalability, and an ability to utilize an append operation [3].

Figure 3 shows a diagram of the file opening process in FusionFS. When a file is opened, FusionFS uses the file path as the key to retrieve the metadata from the distributed hash table. This metadata includes the IP address of the node that stores the file. If the file happens to be on the current node, then the file is simply opened, otherwise the file is retrieved using the file transfer service. FusionFS uses its own file transfer service called Fusion Data Transfer (FDT) on top of UDP-based Data Transfer (UDT). Note that the diagram has been simplified and Node-j does not necessarily contain both the metadata and the file.

FusionFS's unique approach to metadata allows it to eliminate the bottlenecks created by metadata servers and allows FusionFS to scale much better than many other distributed file systems. Benchmarks have shown that FusionFS provides metadata rates and I/O throughput that is almost two orders of magnitude greater than that of GPFS and nearly linear scaling at 1024 nodes[6].

2.3 Optimizing Locality

Our goal is to optimize locality using Swift/T with FusionFS and analyze the performance benefits that are gained. Writes in FusionFS are already local and therefore already optimized. To optimize read performance, our objective is to have tasks scheduled on nodes that contain the data used by those tasks thereby eliminating the cost of transferring files.

Using the language features of Swift/T, we can optimize locality in FusionFS. By informing Swift/T which node to assign to a leaf task, we can reduce the number of data transfers that FusionFs has to perform. The Swift language has an annotation, `@location`, for achieving this kind of control. The location annotation allows the user to specify which node a leaf task should run on. The location annotation can further be customized with soft and hard locations. Soft location tells Swift/T that we prefer this leaf task to run on a particular node, but if that node is busy, perform the task on another node. The hard location specifies that the leaf task must be performed on a particular node.

Optimization in FusionFS was achieved by first creating an application that would connect to the ZHT server and lookup the IP address of a file in the Fusion file system. Then, we used the application to create an app function in Swift called 'lookup'. After converting the IP address to a message passing interface (MPI) rank, we can use the

```
foreach f in fileArray {
    string filename = filename(f);
    string host = lookup(filename);
    location L = ip2rank(host);
    @location=L application(f);
}
```

Figure 4: Pseudo code using location annotation in Swift/T

location annotation to specify the location to perform an application. Figure 4 shows an example of the pseudo code implementing this feature.

3. RESULTS

For all of our tests, we used m3.xlarge EC2 instances. These nodes have 4 cores, 15 GB of RAM and a 40 GB SSD instance store. We first compared the FusionFS write to a write on the local disk. Figure 5 shows the throughput results of 64 by 1GB file writes with one worker per node. FusionFS scaled well in this test with only a 50% decrease in write performance at 60 nodes. Next, we compared read performance of FusionFs with locality to FusionFS without locality. Figure 6 shows the results of 64 by 1GB file reads with one worker per node. After using the locality feature, our results showed a noticeable increase in performance. Performance was steadily increased by six times with little sign of slowing down. These results suggest that FusionFS with locality would scale to a much larger number of nodes.

Next we tested scaling the number of workers per node. Figure 7 shows the throughput results of 64 by 1GB file writes on 8 nodes when scaling the number of workers per node. FusionFS's performance is mostly maintained during this scaling. Figure 8 shows the throughput results of 64 by 1GB file reads on 8 nodes when scaling the number of workers per node. This results suggests that performance gains are maintained when increasing the number of workers per node.

4. RELATED WORK

Similar work has been done using the Hercules file system. Hercules is an in-memory distributed file system based on Memcache. Using Swift's location annotation with the Hercules file system, researchers were able to obtain substantial improvements in I/O throughput. Using locality with writes in Hercules, write throughput nearly doubled. While locality did not improve read performance in Hercules for tests with one worker per node, read throughput more than doubled with eight workers per node. [2]

5. CONCLUSION

Our results shows integrating Swift-T with distributed file systems is a promising field of research. Future work includes performing tests on larger clusters, comparisons with other file systems like GPFS.

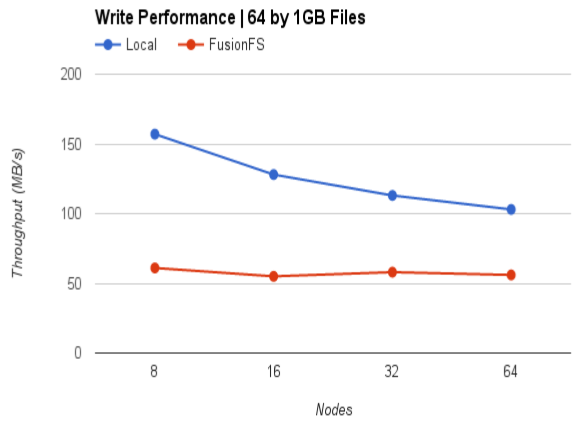

Figure 5: Comparing write performance in FusionFS to Local disk when scaling then number of nodes

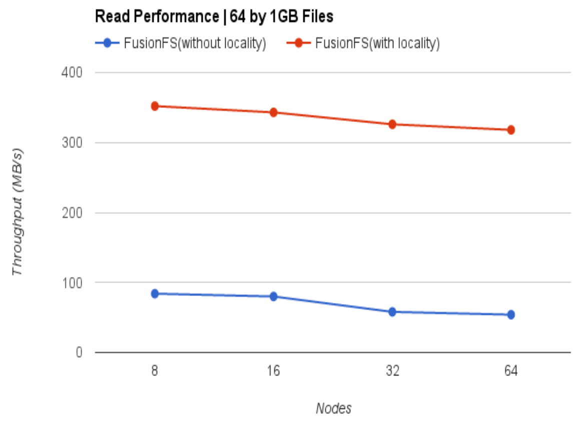

Figure 6: Comparing read performance of FusionFS with locality to FusionFS without locality when scaling the number of nodes.

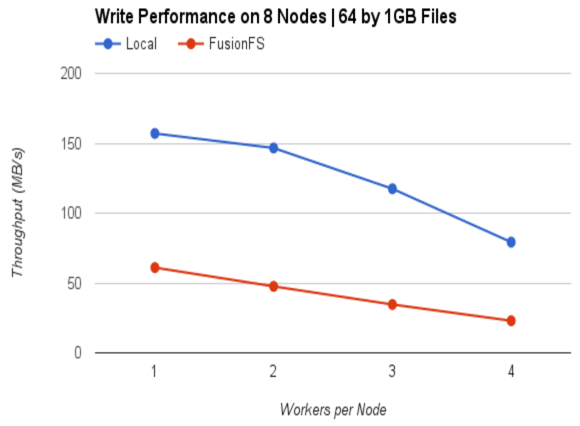

Figure 7: Comparing write performance in FusionFS to Local disk when scaling the number of workers per node.

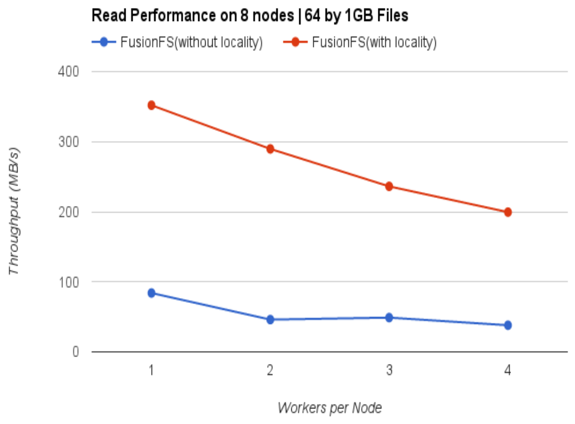

Figure 8: Comparing read performance of FusionFS with locality to FusionFS without locality when scaling then number of workers per node.

6. REFERENCES

[1] T. Armstrong. A dissertation submitted to the faculty of the division of the physical sciences in candidacy for the degree of doctor of philosophy. June 2015.

[2] F. R. Duro, J. G. Blas, F. Isaila, and J. Carretero. Exploiting data locality in swift/t workflows using hercules. In *Proceedings of the Network for Sustainable Ultrascale Computing Workshop*, 2014.

[3] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu. Zht: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 775–787, May 2013.

[4] J. Wozniak, T. Armstrong, M. Wilde, D. Katz, E. Lusk, and I. Foster. Swift/t: Large-scale application composition via distributed-memory dataflow processing. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 95–102, May 2013.

[5] J. M. Wozniak, H. Sharma, T. G. Armstrong, M. Wilde, J. D. Almer, and I. Foster. Big data staging with mpi-io for interactive x-ray science. In *Proceedings of the 2014 IEEE/ACM International Symposium on Big Data Computing*, BDC '14, pages 26–34, Washington, DC, USA, 2014. IEEE Computer Society.

[6] D. Zhao, Z. Zhang, X. Zhou, T. Li, D. Kimpe, P. H. Carns, R. B. Ross, and I. Raicu. Fusionfs: Towards supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *2014 IEEE international Conference on Big Data*, 2014.