Provenance-Based Prediction Scheme for Object Storage System in HPC

Dong Dai[1], Yong Chen[1], Dries Kimpe[2], and Rob Ross[2]

[1]Computer Science Department, Texas Tech University
[1]dong.dai@ttu.edu, yong.chen@ttu.edu
[2]Mathematics and Computer Science Division, Argonne National Laboratory,
[2]dkimpe@mcs.anl.gov, rross@mcs.anl.gov

I. Introduction

Provenance, also known as lineage, is metadata that describes the history of an object [1], [2]. It reveals the detailed information about applications and data sets, which can be used to capture the system status. However, although there have been numerous research studies about the design and implementation of provenance systems [1], [3], to the best of our knowledge, there is no any I/O prediction system that uses such provenance information.

In practice, it is common that scientists re-run applications on the same or different data sets repetitively. In these scenarios, there is a potential to predict the future I/O operations from the historical traces of applications. However, there are two challenges for object storage file systems. First, we need to know whether current execution has exactly shown before or is there any similar executions we can use as history. Second, for applications running on new and different input datasets, we need to identify the possible future inputs before making predictions. Traditionally, this input is given by the file names, but object-based store does not provide such structures that can give an overall sequential order of inputs. In this research, we argue that the provenance is able to provide solutions to address both challenges, which is essential to make an accurate prediction.

In this study, we propose an I/O access prediction system for general object storage by collecting and analyzing provenance. Based on current evaluations on real-world trace data, we have confirmed that provenance analysis provides an accurate I/O prediction with a reasonable complexity.

II. Design and Algorithm

Fig. 1 shows the overall architecture of the proposed prediction system for object-based storage. The prediction system contains components on compute and storage nodes. On compute nodes, a *provenance collector* gathers history of each local execution. Storage APIs are modified with adding provenance to request. On storage nodes, a *provenance builder* is designed to make provenance collector simpler and more efficient for making predictions. A *predictor* generates prediction for each object request.

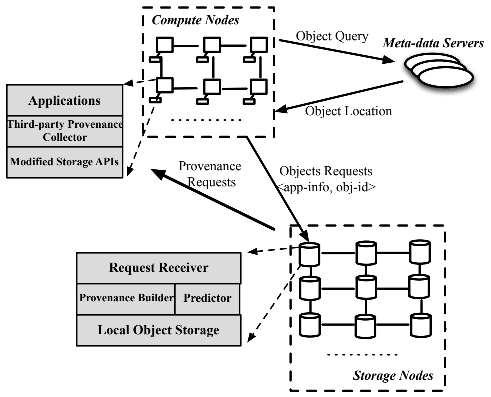

Fig. 1. Overall architecture of provenance-based prediction system for object store

The prediction algorithm works as follows. First, for each request from one execution, it finds out which *similar execution group* that the execution belongs to. If it never happens before, a new group is created and the access pattern is built for this new group based on upcoming I/O accesses. If the current execution belongs to any existing *similar execution group*, then the algorithm obtains possible future objects (*object collection*) from that group and applies access pattern it has built for this group onto the object collection to generate a prediction.

The basic idea of building *similar execution groups* is to cluster executions into different groups based on their distances which are calculated from all variables that are involved during the entire life cycle of an execution. To calculate distances between two executions, we go through all their ancestors and calculate them layer by layer until a pre-defined level. Inside each layer, the distance is the summation of the distances from all their provenance fields. For each provenance field, a difference introduces **1** in distance score. Those distance scores would be adjusted with a weight (W_f) to represent their importance. To find an appropriate W_f that can be used to accurately predict the similarity of two executions, we design a semi-supervised growing clustering algorithm to generate the accurate *execution group* and train W_f. The threshold (thr) is defined as the maximal distance between any two executions in the same group. For each sample, we first find the best collection. If the distance is larger than thr, a new cluster is created for this sample. After a clustering result is achieved from its real I/O accesses, the current W_f is adjusted to force it falls into the right execution group.

After identifying the execution, the algorithm will build its future *objects collection*. The access history of application in object storage can be viewed as an ordered list of objects as Fig. 2 shows. The core idea of building *object collection* is that there will be an application (app_w), running in the system, generated the data for later application (app_r) to visit. We term these two applications *paired applications*. Knowing two applications are paired, we can safely predict the future object collections of the read application based on the outputs of another application.

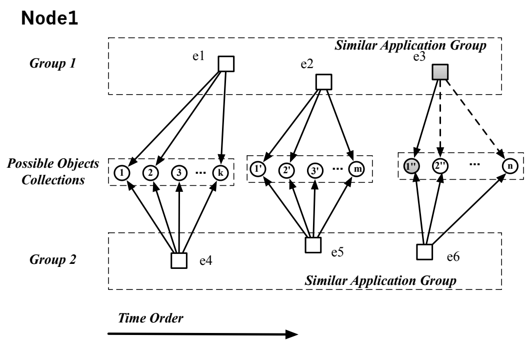
Node1
Group 1
Similar Application Group
e1
e2
e3
Possible Objects Collections
Group 2
e4
e5
e6
Similar Application Group
Time Order

Fig. 2. Build object collection by analyzing provenance information

Fig. 2 demonstrates a paired example for two groups. For each group, we maintain an input object collection and an output object collection. Given a request ($e_3 \rightarrow 1^{''}$), all ancestors of object $1^{''}$ are first retrieved. As can be seen, they belong to group (g_2). The overlap ratio between these two groups is then calculated to detect how many objects read by g_1 are from outputs of g_2:

$$overlap(g_1, g_2) = \frac{|inputs(g_1) \cap outputs(g_2)|}{|outputs(g_2)|} \tag{1}$$

A higher ratio indicates a higher possibility of relevance. If it is larger than the threshold, then these two groups are paired. Thus, the object collection of current execution would be expanded to the output objects collection of e_w in g_w. The object building works only for those object requests that never appear before.

A local pattern detection algorithm adapted from the literature [4] is used to detect the object access pattern inside an object collection. The procedure of applying object access pattern onto the objects collection works as follows: for the current request, if it belongs to current objects collection and we can easily find out current start point in the object access pattern. Then, we generate predictions following the patterns after this start point. For those requests which access objects outside of current object collection, we will update collections by inserting the new objects set, and apply the pattern after the start point onto inserted objects and generate predictions.

III. Evaluation

The current evaluation is conducted based on the simulation on the trace data (Darshan Trace) collected from current top-notch high performance cluster (Intrepid). To evaluate the prediction performance, we replayed all the executions in Darshan trace data and made prediction during read operations. As current trace data does not contain provenance information, we manually provide provenance information for each execution and force our prediction system to learn from the provenance.

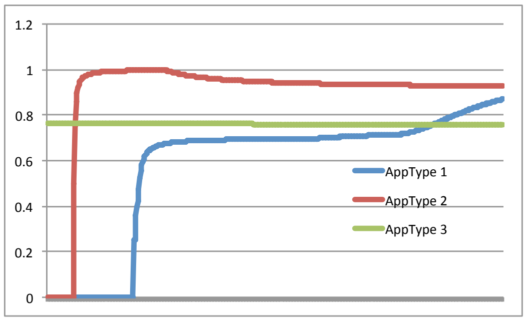

Fig. 3. Prediction accuracy of different application

Fig. 3 shows the prediction accuracy for three different types of applications. The x-axis denotes the whole lifetime of an execution and we normalized them based on the shortest one. In Fig. 3, *apptype 1* denotes similar executions that run on changing data sets, and they were changed by other executions. As the chart shows, the prediction system gave more and more accurate predictions while time is lapsing. *apptype 2* shows applications that reads new inputs. Being different from *apptype 1*, these new inputs were not generated by other executions. We can observe that the accuracy is initially high, but is gradually reduced. The reason is that we do not have enough information to predict these totally new inputs. *apptype 3* shows the most repetitive application. As we can see, it achieved a nearly constant prediction accuracy (80%). In summary, our current evaluations have shown high prediction accuracy with provenance-based predictions.

IV. Conclusion

In this study, we designed and prototyped a novel I/O prediction system for object storage based on provenance analysis. The preliminary evaluations showed its effectiveness, and especially the potential of provenance usage in HPC.

References

[1] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems," in *USENIX Annual Technical Conference, General Track*, pp. 43–56.

[2] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Computing in Science & Engineering*, vol. 10, no. 3, pp. 11–21, 2008.

[3] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *Proceedings of the 13th International Middleware Conference*. Springer-Verlag New York, Inc., pp. 101–120.

[4] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun, "I/o acceleration with pattern detection," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, pp. 25–36.