Guidelines for Client Applications on Avid Unity ISIS

September 2009

Abstract

This document describes the best practices a developer should employ for applications that will run on an Avid Unity ISIS (ISIS) shared storage system. These general guidelines enable applications to extract high performance from ISIS in most usage scenarios.

Unless specifically noted to the contrary, the guidelines described apply to both 1.x and 2.0.x versions of ISIS. Unless noted to the contrary these guidelines apply to both Windows-based and MacOS-based clients.

Introduction

Avid's shared storage systems are designed to support collaborative, real-time media workflows, in which multiple users share common material. ISIS is a unique distributed, parallel file system with advanced file management strategies that deliver on the promise of scalable and predictable performance for workgroups that can scale into the hundreds of users.

Avid Shared Storage products store media in units called "chunks". A chunk is the virtual block size supported by the storage and is the smallest allocation unit. Current versions of ISIS support 256K and 512K chunks. Avid Unity MediaNetwork 5.X storage provides a 1MB chunk.

It is important to remember that ISIS is a high bandwidth and high latency storage system. For an ISIS system there are a number of strategies you should deploy that maximize performance while factoring in the inherent differences between a shared storage system and local storage system:

- Use I/O sizes that balance efficiency and fairness to other clients
- Request data in chunks optimally relative to the physical disk layout, eliminating inefficient read-modify-write I/O's and avoiding crossing physical disk boundaries
- Have client applications use non-blocking I/O calls and queue adequate numbers of these to ensure maximum efficiency of the client/storage connection
- Use a "just-in-time" approach to opening and closing files. It is not optimal to open all of the files required for a multimedia project when the project is loaded. Deferring opens to closer to when data is required (read or write) helps balance the load on the System Director.

By deploying these strategies applications will benefit in a number of ways:

- Faster start when starting media playback
- Higher resolutions and more streams of video per client
- More clients per ISIS system

Guidelines for ISIS Client applications

- Use chunk aligned, chunk sized asynchronous reads for high read performance. In general, the read bandwidth increases with the number of asynchronous reads. If asynchronous reads are not feasible, then one should use large reads – of the order of 16 to 32 MB.[1] These large reads should be multiples of the chunk size and they should be aligned to a chunk. Alignment to a chunk can be achieved by ensuring that the offset of an operation is a multiple of the chunk size. An application should only read what it needs. There is no benefit in reading the trailing parts of a chunk if that data is never used or modified. In ISIS 2.0.x the default chunk size is 256 KB for I1000 and I500 storage blades and 512 KB for I2000 blades.
- Use large writes to achieve high write bandwidth. These writes should be multiples of the chunk size and they should be aligned at a chunk boundary.
- Don't rely on file system caching to cover inefficient read/write operations. A significant issue is when client applications consider the file system cache as an extension of application memory. While this might work on local storage, it does NOT work on shared storage. Since an application cannot be guaranteed that data will be the same next time it is read (remember it's shared) caching time on the client is very limited by design. In most cases, if you read the same data twice through file system calls, the data will likely have to be reread from the disk drive(s) and flow through the storage system twice, and the application will consume more bandwidth from the storage system than necessary. Applications should avoid reading data more than once (data should be cached externally from the file system if it will be re-read soon). One example of inefficient operations is re-reading frames in a slow motion effect rather than caching externally for redisplay.
- Use Direct I/O operations. Using buffered I/O operations decreases throughput considerably by adding memory copies in cache where they are not required.
- When specifying access and share access during file open, specify only what you need. In particular, don't open files for shared write access gratuitously. This indicates to ISIS that the application expects multiple applications to be writing to the file at the same time and may cause ISIS to use aggressive internal flushing to ensure coherency. This can cause performance degradation. Similarly, open files requesting write access only when the application truly intends to write to the file.

Summary

ISIS is a very powerful real-time storage system enabling hundreds of clients to share common media frame accurately. Avid's entire suite of client applications are designed to operate optimally with ISIS, ensuring the full potential of ISIS is attainable by Avid's users.

Third party application developers can also optimize applications for maximum performance of the application and the ISIS system. To do this, there are a handful of key guidelines that should be followed:

- Follow alignment, I/O size and asynchrony guidelines described above.
- Don't rely on file system caching to cover inefficient read/write operations.
- Use Direct I/O operations.
- When specifying access and share access during file open, specify only what you need.

[1] In some cases the operating system may break up an application's I/Os.