社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－OS－102（11）
2006／5／13

ストリーム蓄積・配信処理向け SAN ファイルシステム DJMFS の実装

竹内　理
（株）日立製作所システム開発研究所

要旨

高性能な監視映像蓄積・配信サーバのニーズが高まると共に、サーバの蓄積・配信性能不足や、監視向け機能の実現のための開発工数の高さといった課題も顕在化してきている。本研究では、上記課題を解決するため、既存ストリームサーバとHiTactix搭載の外付けI/Oエンジンサーバをクラスタ構成で連携させる監視映像蓄積・配信サーバの構築方式を提案する。さらに、この構築方式で必須となる高いストリームデータの蓄積・配信性能のスケーラビリティを達成できる SAN ファイルシステム（DJMFS）の設計・実装を行う。特に、DJMFS は、データ挿入、ファイルイメージのスナップショット管理の低オーバヘッドでの実現などの既存 SAN ファイルシステムが持つ技術課題の解決を目指した。最後に、DJMFS の一次性能評価を行い、提案したサーバ構築方式により、10.1GB/s のストリームデータの蓄積・配信が達成可能との見通しを得た。

Implementation of DJMFS: SAN file system for a streaming server

Tadashi Takeuchi
Hitachi Systems Development Laboratory

Abstract

As video surveillance servers are wide spread, demands for the solution of the technical obstacles to achieve high server performance at low development costs are increasing. In order to overcome the obstacles, this paper proposes the novel video surveillance server architecture that consists of clustered external I/O engine servers using HiTactix OS and a conventional streaming server. Besides, this paper also design and implement a novel SAN file system (DJMFS) which is indispensable in the proposed architecture. DJMFS can achieve highly scalable read and write I/O performance, because DJMFS has solved the technical problems of conventional SAN file systems, such as lowering overhead of data insert into a file and file image snapshot management. Finally, this paper evaluates the performance of DJMFS and shows that achievement of 10.1GB/s streaming data I/O can be expected by using the proposed architecture and DJMFS.

1．はじめに

近年のブロードバンド網の普及に伴い、公共施設（道路や河川など）や工場プラント等などの監視映像をブロードバンド網経由で監視映像蓄積・配信サーバに配信、当該サーバにて受信した映像をストレージ装置に集約して蓄積する監視サービスのニーズが高まりつつある[1]。従来のようにテープではなく高集積なストレージ装置に監視映像を蓄積することで、映像検索の容易化や映像保存スペースの節約などのメリットが得られ、特に日本国内において注目を浴びている。しかし併せて、監視映像蓄積・配信サーバの蓄積・配信性能不足や、監視向け機能（タイムシフト型配信機能等）の実現のための開発工数の高さなどの課題も顕在化してきている[2]。

本研究では、上記課題を解決するために、まず、外付け I/O エンジンサーバを用いた監視映像蓄積・配信サーバ構築方式を提案する。本方式では、既存ストリームサーバとストリーム専用高性能 OS HiTactix[3, 4]を搭載したアプライアンスサーバ（外付け I/O エンジンサーバ）をクラスタ構成で連携させることで、少ない開発工数での既存映像サーバへの監視向け機能の追加と、高い監視映像蓄積・配信性能の達成の両立を目指す。さらに上記外付け I/O エンジンサーバの実現において必須となるスケーラブルなストリームデータの蓄積・配信性能を達成する SAN ファイルシステム DJMFS（Distributed Journaled Media File System）の設計と実装を行う。最後に、DJMFS を用いたストリームデータの蓄積・配信性能のスケーラビリティに関する一次評価を行い、提案したサーバ構築方式の有効性に関する見通しを立てる。

2. 監視映像蓄積・配信サーバ構築方式

本章では、1節で述べた課題を解決するために著者らが新規に提案した監視映像蓄積・配信サーバの構築方式の概要について述べる。

図 1 に、提案する監視映像蓄積・配信サーバの構築方式の概要を示す。

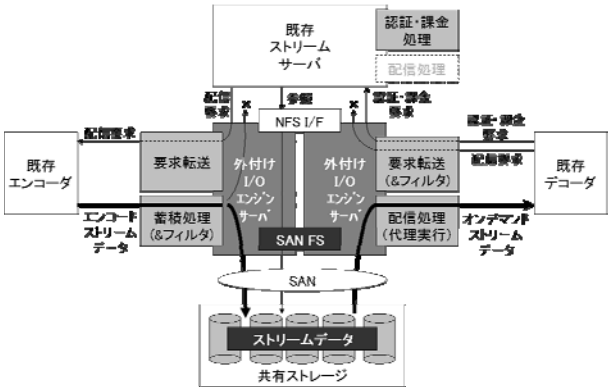

図 1 提案するサーバ構築方式

本方式では、既存ストリームサーバと HiTactix OS を搭載した外付け I/O エンジンサーバが連携する。特に外付け I/O エンジンサーバはクラスタ構成を持ち、スケーラブルなストリームデータの蓄積・配信性能を達成する。

外付けI/Oエンジンサーバは、エンコーダ-既存ストリームサーバ間、及び、既存ストリームサーバ-デコーダ間で送受される要求やストリームデータのフィルタリング機能を持つ。外付けI/Oエンジンサーバは、

- エンコードストリームデータ配信要求
- 認証・課金要求

はそのまま転送するが、

- エンコードストリームデータ
- オンデマンドストリームデータ配信要求

は横取りをし、既存ストリームサーバに転送しない。

さらに、外付けI/Oエンジンサーバは、

- 横取りしたエンコードストリームデータを高性能に共有ストレージ内に蓄積する機能
- 横取りしたオンデマンドストリームデータ配信要求に従い、共有ストレージ内に蓄積されたオンデマンドストリームデータを高性能に配信する機能
- NFS インタフェースを介して蓄積したストリームデータファイルを公開し、既存ストリームサーバにおける認証・課金処理を可能にする機能

を提供する。これらの機能により、既存ストリームサーバに外付け I/O エンジンサーバを追加設置するだけで、ストリームデータの蓄積・配信性能のアクセラレーション、及び監視向け機能(エンコードストリームデータのストレージへの蓄積機能、蓄積されたストリームデータのタイムシフト配信機能)の実現を可能とする。

本方式で構築した監視映像蓄積・配信サーバの蓄積・配信性能のスケーラビリティを確保するためには、蓄積を実行する外付けI/Oエンジンサーバと、配信を実行する外付け I/O エンジンサーバ間でのファイル共有を低オーバヘッドで実現し、かつ、多ストリームの read/write の安定レートでの同時実行を保証できる SAN ファイルシステムが必要となる。次節では、上記実現のために著者らが新規に設計・実装した DJMFS の概要について説明する。

3. DJMFS の概要

本節では、2節で説明したサーバ構築方式において使用するために著者らが新規に設計・実装した DJMFS の概要について説明する。まず、全体構成について述べた後、設計するにあたり解決する必要があった課題とその解決方法について説明する。最後に、実装の概要について述べる。

3. 1. 全体構成

DJMFS は、Tivoli SANergy[5]、SANPoint Direct[6]、CXFS[7]などの従来の SAN ファイルシステムと同様に、共有ストレージに格納されるファイルのメタ情報(ファイルのマップ情報など)を一元管理する DJMFS サーバと、アプリケーションからのファイル I/O 要求を受け付けと、DJMFS サーバからのメタ情報の取得、さらに I/O 先となる共有ストレージのブロック番号の決定と共有ストレージへのブロック I/O を実行する DJMFS クライアントからなる(図2参照)。DJMFS では、同時に複数のアプリケーションが同一ファイルをオープンできる。但し、従来の POSIX 互換のファイルシステムと異なり、write モードでオープンできるアプリケーションは一つのファイルにつき最大で 1 である(ストリームデータの蓄積・配信用途では、この制約は問題にならない)。同一ファイルに対して一つのアプリケーションが write を実行する(以下、write を実行するアプリケーションを「writer」と呼ぶ)と、DJMFS クライアントを介して、適宜

DJMFS サーバにメタ情報の更新が通知される。その結果、同一ファイルをオープンしている他アプリケーション、及び NFS インタフェースを介してアクセスする既存ストリームサーバ(以下「reader」と呼ぶ)からも、この更新結果を参照することができる。

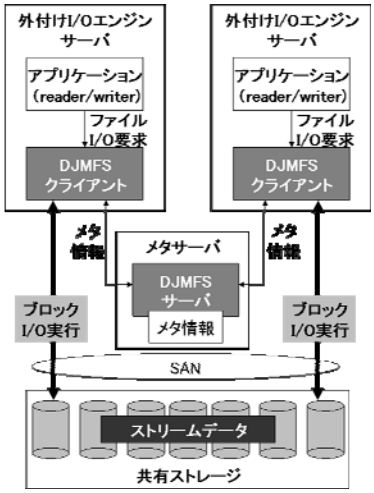

図 2 DJMFS の全体構成

3. 2. 課題

従来の SAN ファイルシステムは、以下の実現が課題となり、ストリームデータの蓄積・配信用途への適用が難しい。

- ストリームデータ挿入の実現
- スナップショット管理の実現
- 排他制御待ちに伴う I/O レート落ち込みの防止

以下では、これらの概要について述べる。

3. 2. 1. ストリームデータの挿入

一般にストリームデータファイルは、ストリームデータを格納する部分と、ストリームデータのメタ情報を格納する部分からなる。例えば、Windows Media サーバ[1]で使用する ASF ファイルフォーマットでは、ヘッダオブジェクト部(ファイルサイズやビットレート等のストリーム属性情報を格納)、データオブジェクト部(実ストリームデータ格納)、インデックスオブジェクト部(再生時刻とデータオブジェクト部格納データのオフセットの関係を格納、ジャンプ再生時に使用)の3つの部分から一つのファイルが構成される。

蓄積が進行するにつれ、これらの各部分のサイズは大きくなりうる。従来の SAN ファイルシステムの API を用いてこの蓄積を実行しようとすると、図 3 に示すように、truncate/append のインタフェースを用いつつ、メモリ-ストレージ間で大きなデータ移動が必要となり、ストレージ負荷が高くなる。ストレージ負荷による蓄積性能のスケーラビリティが喪失を防ぐためには、低オーバヘッドでのファイルへのデータ挿入の実現が必要となる。

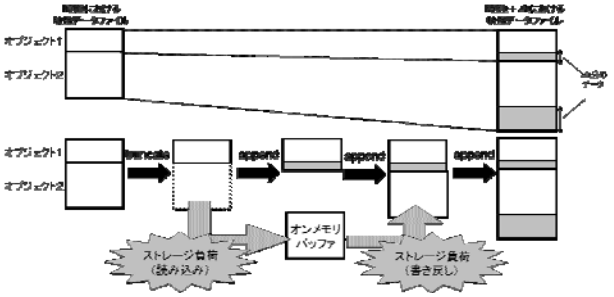

図 3 データ移動の発生

3. 2. 2. スナップショット管理

蓄積の進行とともに、writer はストリームデータの各部分を随時更新する。監視映像蓄積・配信サーバでタイムシフトを実行する場合、同一ファイルを参照しつつ、配信処理を実行する reader も同時に存在しうる。

従来の POSIX セマンティックスを持つ SAN ファイルシステムでは、writer によるすべての更新後のファイルイメージが、reader から参照できる。そのため、一つのストリームデータファイルの、一部は更新済みだか残りは未更新の状態のファイルイメージを reader が参照し、フォーマット異常を検出する可能性がある。これを防ぐためには、writer がファイルイメージのどの時点でのスナップショットを reader に公開するのかを制御する機能が必要になる。

また、2 節のサーバ構築方式を用いてデコーダがストリームデータの配信を受ける際には、図 4 に示す要求・応答の送受が行われる。デコーダによっては、図 4 の☆と★に格納される属性情報(ストリームデータのサイズ等の情報)が不一致の場合に異常を検出する場合がある。この2つを一致させるためには、既存ストリームサーバと外付け I/O エンジンサーバが、応答生成時に同一のスナップショットを参照する必要がある。そのため、図 4 の2つのスナップショットの参照の間では、たとえ writer がスナップショットを更新しても、reader からはその更新が見えない様、スナップショットの更新を一時的にロックする機能も必要となる。上記のようなスナップショット管理を低オーバヘッドで実現す

[1] Windows Media は米国 Microsoft Corporation の登録商標です。

ることが、スケーラブルな蓄積・配信性能の達成のために必要となる。

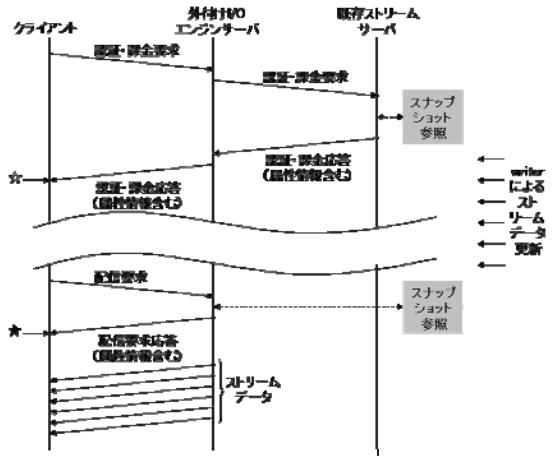

図 4 配信時の要求・応答のフロー

3. 2. 3. I/O レートの落ち込み防止

HiTactix のファイルシステムをはじめ、空きブロックをリスト状に管理するファイルシステムは多い。このようなファイルシステムにおいては、当該リストへのアクセスの際にロック獲得が必要となる。そして、空きブロック確保を要求する複数の writer が同時実行した際には、図 5 に示す通り、大きなロックの獲得待ちが発生しうる。

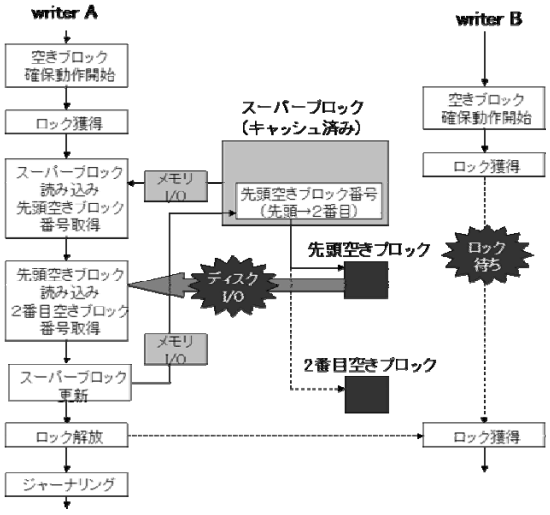

図 5 ロック獲得待ちの発生

図 5 の例では、writer A が、先にロックを獲得、スーパーブロックから接続されている空きブロックリストを更新するため、以下の処理を実行する。

1. スーパーブロックに格納されている先頭空きブロック番号に従い、先頭空きブロックをディスクから読み込む
2. 先頭空きブロックに格納されている2番目の空きブロック番号を読み込む
3. スーパーブロックの先頭空きブロック情報を、2番目の空きブロック番号に更新する
4. スーパーブロック、先頭空きブロックの情報更新をジャーナリングするためのディスク I/O 実行

上記のうち 1～3 は atomic に実行する必要があるため、1 のディスク I/O の実行の間（数十ミリ秒程度）、writer A はロックを保持し続ける。そのため、writer B はその間 write 実行の継続がブロックされ、ストリームデータの書き込みレートが落ち込む。この落ち込みを防ぐには、ロックを獲得しながらのディスク I/O をせずに、空きブロックアクセスの排他制御を実現する必要がある。

3. 3. 課題解決方式

3. 2 節で示した課題を解決するために、DJMFS では

- オブジェクト操作インタフェース
- スナップショットマネージャ
- プリフェッチスレッドを用いた空きブロック管理

を提供する。以下では、これらについて説明する。

3. 3. 1. オブジェクト操作インタフェース

ストレージ負荷をかけることなくデータ挿入を実現するため、DJMFS ではファイルを複数のオブジェクトから構成可能にしている（図 6 参照）。

ファイルへの read 操作は従来のファイルシステムと同様に行われる。しかし、ファイルへの write 操作は、図 6 に示す通りに行われる。

DJMFS では、ファイルへの write インタフェースを呼び出す時に、アプリケーションから、書き込み開始オフセット、サイズ、書き込むデータ以外に、書き込み対象のオブジェクトをも指定可能にしている。そして、書き込み開始オフセット＋サイズがオブジェクト境界を越える際には、次のオブジェクトのデータを上書きせずに、書き込み対象のオブジェクトのサイズを増やす。このインタフェースにより、3. 2. 1節に示すようなデータ移動を行うことなくデータの挿入も可能になる。

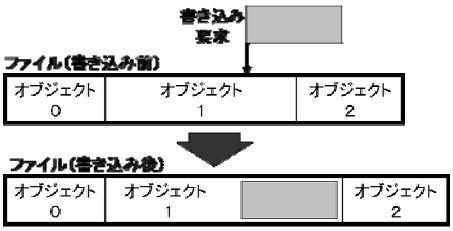

図 6 DJMFS の write 操作

さらに DJMFS は、一つのオブジェクトを従来の一つのファイルのように扱いマップ情報の管理を行う。その結果、たとえ特定のオブジェクトサイズが大きくなっても、ファイルシステム内部で、3．2．1節で示すようなデータ移動を行う必要もない。

上記に示すようなオブジェクト操作を可能とするために、DJMFS は、表1に示す API を提供する。オブジェクトの生成、削除インタフェースの他に、オブジェクトポインタの設定インタフェースを提供し、従来の write インタフェースとの互換性を維持しながらも、アプリケーションが書き込み対象のオブジェクトを指定することを可能にする。

表1 DJMFS のオブジェクト操作インタフェース

インタフェース名	概要
CREATE_OBJ	オブジェクトを生成する
DELTE_OBJ	オブジェクトを削除する
SET_OBJPTR	オブジェクトポインタを設定する
GET_OBJPTR	オブジェクトポインタを取得する

3．3．2．スナップショットマネージャ

DJMFS では、

- writer により更新されるファイルイメージのスナップショットを DJMFS クライアントから受信して保存する機能
- 保存されたスナップショットを取得、DJMFS クライアントに送信し、reader に参照させる機能
- writer によるファイルイメージのスナップショット更新を一定時間ロックする機能

を低オーバヘッドで提供する。特に第1の機能の動作契機をアプリケーションから制御可能にすることで、任意の時点でのスナップショットを writer から reader に公開することを可能にする。これらの機能は、DJMFS サーバ上で動作するスナップショットマネージャが実現する。

スナップショットマネージャは、図 7 に示すデータ構造を用いてスナップショットを管理する。本マネージャは、ストリームデータファイルを構成する各オブジェクトは、

- データサイズが小さく、かつランダム write が発生するオブジェクト
- データサイズが大きいが、append しか発生しないデータオブジェクト

のいずれかであることを利用してスナップショット管理を効率化している。例えば、ASF ファイルフォーマットの場合、ヘッダオブジェクト部とインデックスオブジェクト部は前者になり、データオブジェクトは後者になる。

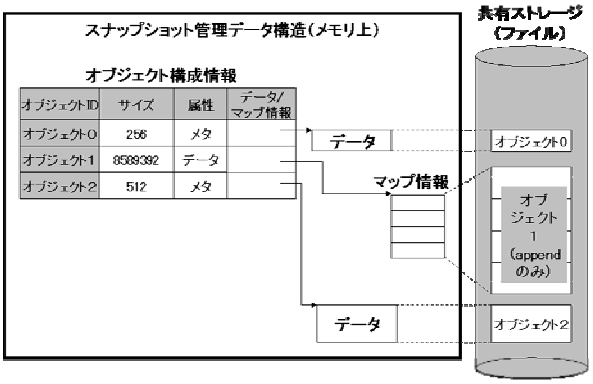

図 7 スナップショットの管理データ構造

前者のオブジェクトは、全データとサイズを任意の時点のスナップショットとして保持する。一方、後者のオブジェクト部は、マップ情報とサイズしかスナップショットとして保持しない。そのため、スナップショットを保存した後に、writer により共有ストレージの内容が更新されると、マップ情報との不一致が発生する。しかし、当該オブジェクトには append 処理しか行われず、上書きが発生しないため、上記更新発生後も、保存した時点でのスナップショットの取得が可能である。

図 7 のデータ構造により、スナップショット管理のために必要なデータ構造を小さく保てる。結果として、DJMFS サーバ-クライアント間で送受すべきデータ量を削減し、スナップ処理管理のオーバヘッドを低減できる。

上記スナップショット管理を実現するため、DJMFS はアプリケーションに、オブジェクトの属性を設定／取得するためのインタフェースを提供する。オブジェクト属性にはメタ属性／データ属性を定義でき、前述したスナップショット管

理のためのデータ構造を変えられる。さらに、データ属性を持つオブジェクトには、SYNCHRONOUS 属性／非 SYNCHRONOUS 属性の設定もできる。SYNCHRONOUS 属性を持てるオブジェクトは、ファイルにつき高々一つである。SYNCHRONOUS 属性を持つオブジェクトに対して write が発生すると、前述したスナップショットの転送、保存が DJMFS クライアント-サーバ間で行われる。writer は、当該オブジェクト更新後のファイルイメージが reader 側に公開されても良いようにプログラミングする必要がある。さらに、DJMFS は、表2に示すスナップショット操作インタフェースも提供し、reader によるスナップショットの取得やロックも可能にしている。

表2 スナップショット操作インタフェース

インタフェース名	概要
SYNC_SNAP	スナップショットを取得する
LOCK_SNAP	UNLOCK_SNAP までスナップショット更新をロックする
UNLOCK_SNAP	スナップショット更新ロックを解除する

3．3．3．プリフェッチスレッド

3．2．3節で述べたように、DJMFS では、ロックを保持したままのディスクI/O 実行を防ぎつつ、空きブロックリストの排他制御を実現している。空きブロックリスト管理は、以下のアルゴリズムにより行っている。

1. スーパーブロックに格納されている先頭空きブロックから一定数の空きブロックを、予めディスクから先読みし、メモリ上にキャッシングするプリフェッチャスレッドを動作させる。併せてキャッシングされている空きブロック数と、次にプリフェッチすべき空きブロックリストへのポインタをスーパーブロック内のデータ構造に追加する。
2. 空きブロックを獲得する際には、従来通りロックを保持しながら、スーパーブロックや先頭空きブロックの情報を更新する。しかし、空きブロックのプリフェッチスピードが、空きブロックの要求スピードより大きければ、先頭空きブロックは既にメモリ上にキャッシングされているため、write 実行スレッドがロック獲得中にディスク I/O を実行することはない。
3. プリフェッチャスレッドは、ロックを獲得後、キャッシングされている空きブロック数を検査。当該空きブロック数が 1 以上であれば、ロックを直ちに解放して、スーパーブロックに格納されている次にプリフェッチを行うべき空きブロック番号より、プリフェッチ数が一定数に達するまでプリフェッチ動作を継続する。上記空きブロック数が 1 以上であれば、プリフェッチャスレッドが空きブロック獲得スレッドと同一空きブロックを参照する可能性はないため（図 8 参照）、直ちにロックを解放しても空きブロックリストの一貫性は失われない。空きブロック数が 0 であれば、当該空きブロック数が 1 以上になるまでロックを獲得し、プリフェッチ動作を実行する。2 と同様、空きブロックのプリフェッチスピードが、空きブロックの要求スピードより大きければ、当該空きブロック数は 1 以上になるため、プリフェッチャスレッドがロック獲得中にディスク I/O を実行することはない。

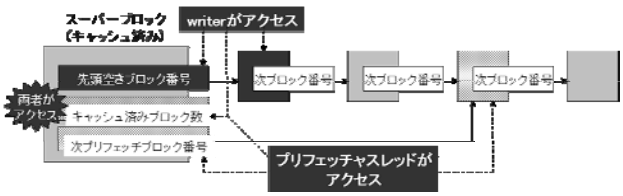

図 8 空きブロックリストのアクセス範囲

3．4．実装

著者らは、HiTactix 上に DJMFS のプロトタイプを実装した。本プロトタイプでは、実装を容易にするため、HiTactix 上に実装されたジャーナルドファイルシステム（JMFS）の機能を一部拡張し、その上に DJMFS を実装している。図 9 に DJMFS のプロトタイプシステムの構成を示す。

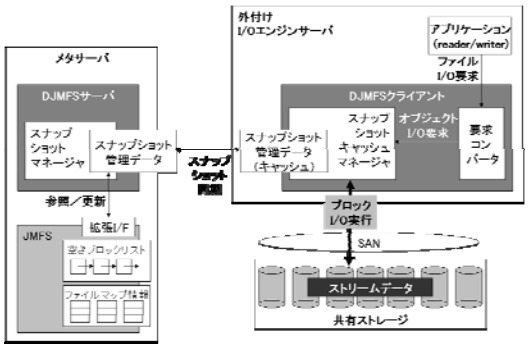

図 9 プロトタイプシステムの構成

DJMFSのファイルを構成する各オブジェクトは、JMFSのファイルとして管理されている。DJMFS 層では、各オブジェクトのスナップショット管理と、アプリケーションからのファイル I/O 要求をオブジェクトへの I/O 要求(及びブロック I/O 要求)に変換する処理のみを行う。

DJMFS層のスナップショットマネージャによるJMFSファイル(DJMFS オブジェクト)マップ情報管理を実現するため、JMFS に以下の拡張インタフェースを追加した。

- 指定ファイル(オブジェクト)のマップ情報を取得／設定するインタフェース
- 空きブロックの確保／解放するインタフェース

DJMFS 層におけるファイルシステムの一貫性チェックを実現するため、

- オブジェクトディスクリプタ一貫性保持機能
- 空きブロック回収機能

を実装した。

オブジェクトディスクリプタ一貫性保持機能では、スナップショットマネージャが管理するオブジェクトディスクリプタと各オブジェクト(JMFS ファイル)との間の一貫性保持を実現する。スナップショットマネージャは、ファイルオープン要求が到達する度に、両者の一貫性チェックを行う。もし一貫性喪失を検出すると、当該ファイルを削除すると共に、ファイルオープン要求のエラーリターンを行う。

空きブロック回収機能は、空きブロックの確保要求を JMFS 層に発行したが、当該ブロックを含むマップ情報の設定や、当該ブロックの解放要求を発行する前に DJMFS サーバがクラッシュした場合に、当該ブロックがファイルシステムから再利用できなくなることを防止する。毎回ブート時に空きブロックリストにも、いかなるファイルのマップ情報にも登録されていないブロックが存在するかチェックし、もし存在すれば空きブロックに再登録する。

4. 性能評価

DJMFS の蓄積・配信性能スケーラビリティの定量的な一次評価を行い、2節で提案した監視映像蓄積・配信サーバの構築方式の有効性に関する見通しを立てた。本節では、まず行った性能評価実験の概要について説明した後、その評価結果について述べる。

4. 1. 評価実験

評価実験システムを図 10 に示す[2]。本実験システムでは、DJMFS サーバ（以下、「ロック管理サーバ」と呼ぶ）搭載マシン 1 台、DJMFS クライアント搭載マシン 2 台、共有ストレージを接続し、各 DJMFS クライアント上で、256KB/s のレート（64KB の I/O サイズ、256ms 周期）にて同一ファイルの read 及び write を同時実行する reader/writer の組を、可変個ずつ並列動作させた。2 台の DJMFS クライアント搭載マシンのうち 1 台は reader のみを、もう 1 台は writer のみを動作させた(以下、それぞれ「reader 実行サーバ」、「writer 実行サーバ」と呼ぶ)。reader/writer の組数を変化させた際の、

- ロック管理サーバマシンの CPU 負荷
- reader/writer におけるデッドラインミス発生率（I/O 要求を発行してから 1 周期すなわち 256ms 後に I/O が完了していない割合）
- ロックビジー率

を測定し、DJMFS の蓄積・配信性能のスケーラビリティを評価した。同様の実験を Fedora Core 4 上に実装された GFS 6 .0[8]（分散ロックプロトコルは DLM を使用[3]）を用いて行い、DJMFS の場合と比較した。

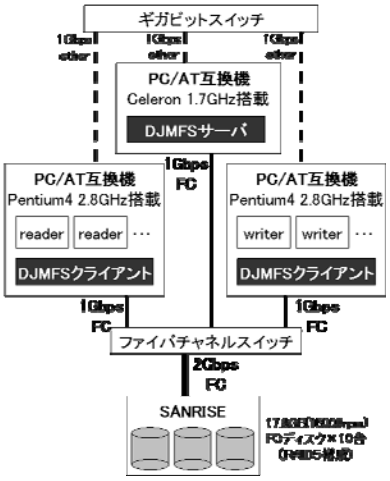

図 10 評価実験システム

[2] Celeron, Pentium は米国 Intel 社の登録商標です。

[3] DLM の場合、writer 実行サーバ上でロックも管理する。そこで、reader 実行サーバと writer 実行サーバの 2 台を準備、両者を同時に動かした場合と後者のみを単独で動かした場合の writer 実行サーバの CPU 負荷の差を算出し、ロック管理サーバの CPU 負荷とした。

4. 2. 評価結果

実験結果を図 11 に示す。

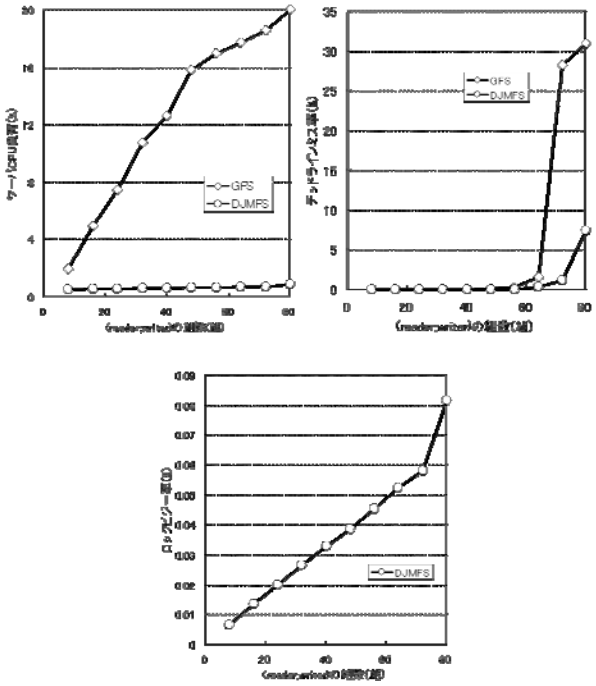

図 11 実験結果

実験システムでは、共有ストレージ性能がボトルネックとなり、80 組（160 ストリーム）以上の reader/writer の同時実行が行えなかった。しかし、共有ストレージ性能が無限だと仮定すると、DJMFS を用いた場合、20,000 組（40,000 ストリーム）の reader/writer の同時実行（10.1GB/s の read/write の同時実行）を行っても、ロック管理サーバ負荷は 100%に達成しないと予測できる。一方、GFS の場合は約 293 組（586 ストリーム）の reader/writer の同時実行でロック管理サーバ負荷が飽和することが予測できる。すなわち、DJMFS は GFS と比して、蓄積・配信性能のスケーラビリティを約 34 倍向上させている。HiTactix エンジンサーバ 1 台あたり約 200MB/s の蓄積・配信性能を持つ[3]ことから、2 節のサーバ構築方式を用いれば、ロック管理サーバを飽和させずに、50 台の外付け I/O エンジンサーバが並列動作可能であると期待できる。

また、DJMFS を用いた場合、reader/writer の組を 20,000 組並列動作させた場合のロックビジー率は約 18.3%と予測でき、ロック獲得待ちによる I/O レートの大きな落ち込みも発生しないことを期待できる。

5. まとめ

本研究では、監視映像蓄積・配信サーバの蓄積・配信性能不足や、監視向け機能の実現のための開発工数の高さといった課題を解決することを目的に、監視映像蓄積・配信サーバの構築方式を提案した。さらに、この構築方式で必須となるスケーラブルな蓄積・配信性能を達成可能な SAN ファイルシステムである DJMFS の設計・実装を行った。最後に、DJMFS のスケーラビリティの評価を行い、共有ストレージの性能ボトルネックが解消されれば、最大 50 台のサーバが並列動作し、10.1GB/s のストリームデータの蓄積・配信を実行する監視映像蓄積・配信サーバが実現可能との見通しを得た。

参考文献

[1] 山崎洋一,「実践 e-Japan 追跡・政府プロジェクト」, 日経コミュニケーションズ, pp110, Jun. 2004.

[2] 川村智,「映像蓄積配信システム HEC21/VS」, 日立エンジニアリング（株）ザ・ベストソリューション, Vol.2, pp94-101, Mar. 2005.

[3] D. Le Moal, et. al, “Cost-Effective Streaming Server Implementation Using Hi-Tactix”, Proceedings of the 10th ASM International Conference on Multimedia, pp382-391, Dec. 2002.

[4] 竹内理他,「外付け I/O エンジンを用いたストリームサーバの実現」, 情報処理学会論文誌, Vol.43, No.1, pp137- 145, Jan. 2002.

[5] C. Brooks et. al., “A Paractical Gude to Tivoli SANergy IBM Redbook SG246146”, http://www.redbooks.ibm.com/, Jun. 2001.

[6] VERITAS Software Corporation, “VERITAS SANPoint Direct File Access”, White paper, Aug. 2000.

[7] Silicon Graphics, “SGI CXS Clustered File System”, Datasheet.

[8] K. W. Preslan, et. al., “Implementing Journaling in a Linux Shared Disk File System”, 17th IEEE Symposium on Mass Storage Systems, pp351-378, Mar. 2000.