情報処理学会論文誌　コンピューティングシステム　Vol.5　No.3　42–56 (May 2012)

ユーザによる明示的な予約に基づきI/O性能を保証する分散ストレージシステム

谷村 勇輔[1,a)]　鯉江 英隆[1,2]　工藤 知宏[1]　小島 功[1]　田中 良夫[1]

受付日 2011年10月7日, 採録日 2012年2月5日

概要：クラウドやグリッドでは，ネットワーク越しに提供されるサービスの性能保証への期待が大きい．特に，課金を行うサービス提供者はSLO（Service Level Objective）を明確にし，ユーザとSLA（Service Level Agreement）を締結することになる．しかし，計算機やネットワークの性能保証は得られてもストレージの性能はベストエフォートを前提にせざるをえないという問題がある．本論文では，ユーザが明示的に時間を指定してI/O性能とストレージ容量の予約を行うことにより，ストレージへのアクセス性能を保証する分散ストレージシステムを提案する．これは予約に基づいてディスクやI/Oパスの使用権を割り当て，予約時間帯にストレージデバイスからアクセス・クライアントまでのI/O制御を適切に行うことで，要求されたI/O性能を提供するものである．この仕組みを実装したプロトタイプを用いた評価実験により，Readに関して，予約に基づく性能保証がストレージシステムへの同時アクセス発生時に有効であることを示した．また，Web Servicesに基づくインタフェースを提供することでストレージへのアクセスとネットワークの帯域，計算機の同時予約が容易になることと，同時予約を使った応用例を提示して提案システムの有用性を示した．

キーワード：性能予約，性能保証，資源管理，分散ストレージ，クラウド

A Distributed Storage System Providing I/O Performance Guarantee According to User's Advance Reservation

Yusuke Tanimura[1,a)]　Hidetaka Koie[1,2]　Tomohiro Kudoh[1]　Isao Kojima[1]
Yoshio Tanaka[1]

Received: October 7, 2011, Accepted: February 5, 2012

Abstract: Performance assurance has become an important aspect in cloud and grid computing where services are provided over the Internet. In particular, the service providers who charge fees for resource usage define Service Level Objective (SLO) and deal with Service Level Agreements (SLA) with users. However, the I/O performance of the storage or data access service is still provided on a best effort basis while computing speed and network bandwidth are often guaranteed. This paper proposes a distributed storage system which guarantees I/O performance to users, by allowing application users to explicitly make an advance reservation for I/O access and storage space, with start and end time of their use. In our proposed architecture, the storage system manages use of disks, I/O paths and etc. based on the reservation, and enforces I/O priorities on the end-to-end path from storage devices to access clients during the reserved time, in order to satisfy the performance requirement. The evaluation result using a prototype implementation indicates that the requested read performance was achieved by this mechanism even when multiple accesses are issued to the storage system simultaneously. In addition, the paper presents that the storage performance can be easily reserved with network bandwidth and computing speed by providing a Web Services based reservation interface, and introduces application use cases of such a combined reservation.

Keywords: performance reservation, performance guarantee, resource management, distributed storage, cloud computing

1. はじめに

インターネットの初期の頃より，インターネット越しに提供されるサービスの性能はベストエフォートが前提となっていた．これはインターネットがベストエフォート網であり，サービス側で個々のユーザに対して性能を保証してもほとんど意味がなかったためである．しかし，ネットワーク技術が進化し，今日では Lambda パスのように広帯域のネットワークをアプリケーションごとに動的にセットアップすることが現実的になってきた [1], [2] ことで，サービス側が性能品質を保証することが重要になってきた．実際，クラウドやグリッドにおけるビジネス・アプリケーションでは，サービス提供者は SLO（Service Level Objective）の中で提供する性能品質を明確にしつつある．SLO を達成するためには，データセンタ内の計算機やストレージ，ネットワークを適切に管理し，それぞれのアプリケーションが要求する性能を満たすような割当てを行う技術が必要である．計算機に関してはバッチスケジューラ，ネットワークに関しては帯域予約によりそうした管理が可能であるが，既存のストレージシステムには類似機能がなく，個々のアクセスに対して完全な性能保証を行うことが困難である．ストレージはデータ共有のために複数のアプリケーションによって共用されることが多く，アクセスが競合して I/O 能力の限界に達する可能性があるが，その場合には個々のアクセスは必要な性能を得ることができない．

本研究ではクラウドやグリッドで用いられている高スループットかつ大容量を提供する分散ストレージシステムを対象にアクセス時の I/O 性能を保証する仕組みについて検討した．分散ストレージシステムは通常 1 つ以上のディスクドライブを備えた複数のストレージサーバからなり，ディスク容量を統合するだけでなく，並列 I/O により高いスループットを提供するという特長がある．つまり，分散ストレージシステムにおいて性能保証のための優先 I/O 制御を行うには，ディスクや I/O パス，サーバキャッシュなどをストレージ資源として管理し，個々のアクセスに割り当てる必要がある．Bigelow らは End-to-End で性能保証を行うための分散ストレージシステムのアーキテクチャを示している [3] が，具体的な資源予約や割当ての手順については言及していない．一方，優先 I/O 制御を行える既存のディスク I/O スケジューリング [4], [5], [6] は個々のアクセスに対してアクセス要求の到達順に資源を割り当てる．したがって，遅れてアクセスを開始するアプリケーションは，資源が不足する状況において必要な性能を享受できないという問題がある．

本論文は，アクセス負荷に対して受動的に優先 I/O 制御を行うのではなく，アプリケーション・ユーザが明示的にストレージシステムに対してアクセス時間を指定して性能予約を行えるようにすることで，この問題を解決するアプローチを提案する．提案アプローチでは実行時にアクセスが拒否されたり性能が制限されたりするのではなく，ユーザは予約の段階で得られる性能を確定することができ，資源が不足する状況においてはアクセスを行うアプリケーションの実行時刻の変更を検討することも可能である．予約が成立すると，ストレージシステムは性能要求に基づいて資源を割り当て，予約時間帯に優先的に I/O 要求を処理することで，アクセスの集中や競合による性能低下を抑制し，ユーザが指定した性能を提供する．単一のディスク性能を上回る性能が要求された場合には，ストレージシステムは内部的にストライピングの利用を決定し，並列にアクセスする複数のディスクをその要求に対して割り当てる．

一方，提案アプローチの検討を進めるうえで End-to-End での I/O 性能制御を実現する必要がある．しかし，ストレージの性能は多くの場合，アクセスパターンに依存し，あらゆるアクセスパターンに対応して性能制御を行うのは容易ではない．そこで，本研究では高スループットを要求するアプリケーションに応用を絞り，open から close までの 1 つのアクセス内では Read と Write の混在がない逐次アクセスにアクセスパターンを限定し，指定された目標スループット（単位時間あたりに Read または Write 処理されるデータ量）を提供する I/O 性能制御を実現することとした．そして，ストレージデバイスに Solid State Drive（SSD）を用い，Read アクセスに関して予約に基づいた I/O 性能制御を行う機能を実装したプロトタイプを開発し，提案アプローチの基本的な概念とそれを実現するアーキテクチャの有効性を評価した．本論文で述べる本研究の成果は以下の 3 点である．

- 予約に基づいて性能を保証する分散ストレージシステムの概念とそれを実現するアーキテクチャを提案した．
- ネットワークの帯域制御と SSD を用いたディスク I/O スケジューリングを統合し，予約に基づいて End-to-End で Read アクセスの性能（スループット）を確保する仕組みをプロトタイプに実装し，評価実験を通して提案アーキテクチャの有効性を示した．
- Web Services に基づくストレージの性能予約インタフェースを開発して，ストレージ性能とネットワーク帯域の同時予約を可能にし，映像配信のデモ実験およびその他の応用例を提示して上記の性能予約の概念の有用性を示した．

以降では，まず 2 章で本研究が対象とするストレージアクセスと性能予約において保証する性能指標，それを実現する提案アーキテクチャについて述べる．そして，性能保

[1] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Ibaraki 305–8568, Japan
[2] 数理技研
SURIGIKEN Co., Ltd., Shinagawa, Tokyo 141–0022, Japan
[a)] yusuke.tanimura@aist.go.jp

証のために必要な予約インタフェース，クライアント API，ストレージ資源管理，I/O 性能制御フレームワークの各コンポーネントの詳細を述べる．次に 3 章で貪欲法を用いた資源割当て手法，ネットワークとディスク I/O スケジューリングの 2 つの I/O 制御手法を組み込んだプロトタイプシステムの実装について述べ，4 章においてそれを用いた評価実験の結果を示す．さらに 5 章では提案ストレージシステムが対象とする高スループットを要求する実アプリケーションの例や他の IT 資源との同時予約を用いた事例を紹介する．6 章で関連研究について述べた後，最後にまとめと今後の課題について述べる．

2. 予約に基づいて動作する提案ストレージシステム

2.1 予約対象とストレージインタフェースの設計

ストレージアクセスの性能予約を実現するには，ストレージの性能がアクセスパターンに依存する問題とアプリケーションが要求する性能の表現方法を考えなければならない．しかし，あらゆるアクセスパターンに対応して性能保証を行うのは容易でないため，本研究ではクラウドやグリッドにおいて高スループットを提供するストレージサービスに応用を絞った．その主要なアクセスパターンはサイズの大きいファイルに対する逐次的なアクセスであり，その性能指標は「I/O スループット（単位時間あたりにストレージシステムにおいて Read または Write 処理されるデータ量であり，単位は “bytes/sec”．以降では単にスループットと記述）」である．そこで，提案ストレージシステムでは性能予約としてスループットの予約をサポートすることとし，予約時間帯においてストレージシステムが推奨する一定の I/O サイズを用いて連続的にアクセスが行われる場合にスループットを保証する設計とした．実装を容易にするために最初は逐次アクセスのみを考えて，open から close までの 1 つのアクセスにおいて Read と Write の混在は認めず，アクセスモードは “create”，“read-only”，“append-open” のみとする．また，保証するスループットの計算に用いる単位時間は，性能保証の粒度としてユーザとストレージシステムの間で交渉可能な仕組みを用意する．

一方，予約に基づいた Write アクセスがディスクスペースの不足のためにエラーとなる状況を避けるためには，性能予約と合わせたディスクスペースの予約が必要である．そこで，クラウドのストレージサービスとして広く知られている Amazon S3 [7] で使われているバケット/オブジェクトのセマンティクスを導入し，スペース予約と組み合わせて提案ストレージシステムのインタフェースを設計した．バケットはオブジェクトを格納する入れ物であるが，提案ストレージシステムではユーザのプライベートなディスクスペース，すなわちスペース予約とも 1 対 1 に対応する．ユーザはバケット名と合わせてスペース・サイズを指定してバケットを作成することでスペース予約が行える．さらに，バケットの作成時には，そのバケットにアクセスする際に保証してほしいスループットとバケットの生存期間を指定可能とした．保証スループットはバケットに割り当てるストレージ領域をいくつの，あるいはどのストレージデバイスを用いて提供するのかを決定するのに用いられ，別途行うことになる性能予約では，この保証スループットを超えない範囲での予約をサポートする．また，スループットだけでなく，ストレージの容量も時間に基づいて予約管理可能な「ストレージ資源」として扱う方針を採用し，全バケットに生存期間を持たせる設計とした．ディスクスペースの効率的な利用を促進するため，生存期間を超えるとバケットとその中に保存されたオブジェクトは提案ストレージシステムによる自動削除の対象になる．

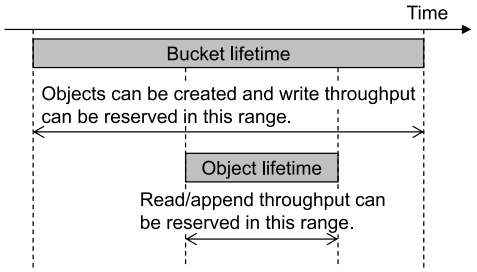

図 1　バケットとオブジェクトの生存期間とそれぞれに対して行える予約

Fig. 1　Lifetime and reservation types for buckets and objects.

主機能である性能予約はバケットとオブジェクトのいずれに対しても行えるものとし，予約時にはバケットのスループットの上限を超えない範囲のスループットとアクセス時間を指定する設計とした．バケットに対する性能予約はオブジェクトを作成するための Write だけであるのに対し，オブジェクトに対する性能予約は Read または Append の 2 種類が考えられる．**図 1** はバケットとオブジェクトのそれぞれの生存期間，行うことのできる予約の種類を示している．性能予約とスペース予約は別々に行うことも可能であるが，同時に行うことも可能である．たとえば，スペース予約とそのスペース上に作られる複数のオブジェクトの性能予約を組み合わせた予約を行うことができる．

提案ストレージシステムでは成立した予約のキャンセルもサポートする．ただし，実装を容易にするため，予約のキャンセルに対しては次の単純なルールを適用する設計とした．性能予約の有無にかかわらず，バケットやオブジェクトは削除可能であり，削除と同時に関連する性能予約は自動的にキャンセルされる．ただし，バケット内にオブジェクトが存在する場合，バケットの削除（スペース予約のキャンセル）はできず，オブジェクトの削除を先に行うことをユーザに求める．

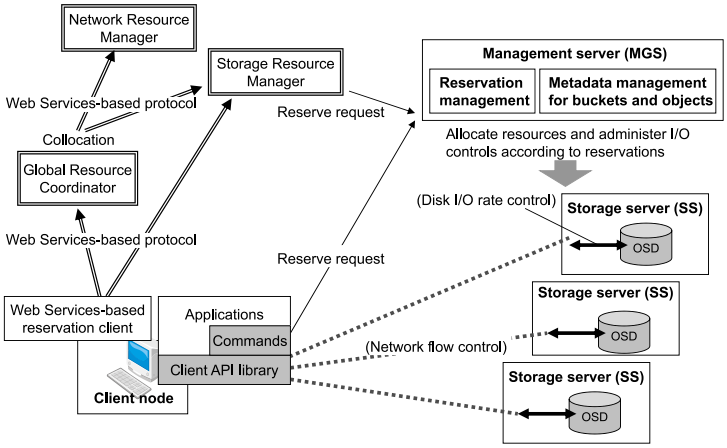

図 2　提案ストレージシステムのアーキテクチャ概要
Fig. 2　Overview of the architecture.

2.2　アーキテクチャ概要

本研究では前節で述べた予約とそれに基づいて性能保証を実現するストレージシステムとして，**図 2** に示すようなアーキテクチャを設計した．これは以下に述べる 2 種類のサーバとクライアント・ツールからなり，それらは TCP/IP ネットワークまたは SAN（Storage Area Network）ベースのネットワークで接続されている．

Management Server（MGS）：MGS はバケットとオブジェクトのメタ情報や名前空間の管理，各 I/O に関与するストレージ資源の情報，システムが受け付けた予約情報などを管理するサーバである．ユーザから見て，MGS は提案ストレージシステム内において 1 つだけ存在する．

Storage Server（SS）：SS は実際の I/O 処理を行うサーバである．SS は提案ストレージシステム内において 1 つ以上存在し，それぞれ 1 つ以上のストレージデバイス（ハードディスクドライブなど）を有する．SS のストレージデバイスはオブジェクトベース・ストレージデバイス（Object-based Storage Device：OSD）として抽象化し，ディスクスペースの割当てや予約は OSD の内部で管理する．クライアントからの I/O 要求は SS が受け付けて，I/O 要求の一部に含まれる MGS の命令に従って OSD に指令を出し，I/O の優先処理やセキュリティ制御を実施する．一方，MGS と SS の間では OSD に対するスペース予約，OSD のモニタリング情報などがやりとりされる．

Client：提案ストレージシステムでは，予約およびデータアクセス用にクライアント API ライブラリとユーティリティ・コマンド群をユーザ向けに提供する．API ライブラリはストライピングを用いた並列 I/O を内部的にサポートする．ストライピングの利用の有無は MGS が内部的に判断し，クライアントライブラリに伝えられる．MGS による判断は，要求性能を満たすために必要な OSD 数の見積りと OSD の空き情報に基づいて行われる．

図 2 の Storage Resource Manager（SRM）は提案ストレージシステムの主要なサービスではないが，標準規格に準じた予約インタフェースを提供する．SRM を用意することで，上位の資源コーディネータは計算機やネットワークのような他の IT 資源の予約と合わせて提案ストレージシステムに対して性能やスペースの予約を行うことができるようになる．

2.3　主要コンポーネント

提案ストレージシステムでは性能予約を実現するために図 2 のアーキテクチャ上に次の 4 つのコンポーネントを用意する．以下ではコンポーネント別にその詳細を述べる．

2.3.1　予約インタフェース

提案ストレージシステムはコマンドライン・インタフェースと Web Services に基づくインタフェースの 2 種類を提供する．ユーザはこれらを用いて性能とスペースの予約，修正，キャンセルを行うことができる．Web Services に基づくインタフェースは，G-lambda プロジェクト [2] によって作成された GNS-WSI3 仕様 [8] を用いている．GNS-WSI3 仕様はネットワーク資源（たとえば，Lambda パス）の予約のために作られたポーリングベースの予約プロトコルであるが，新しい資源タイプを定義することで，予約手順を変更することなくストレージ資源を含むその他の IT 資源の予約に適用可能である．GNS-WSI3 仕様は GridARS [9] によって実装されており，GridARS を用いることで GNS-WSI3 仕様に準拠した予約インタフェースを備えた Storage Resource Manager（SRM）を容易に開発できる．

図 3 は 2.1 節で述べた予約に対応するために新しく定義したストレージ資源のデータ型であり，SRM に送る予約要求を記述するのに用いられる．ServicePoint は，その要求を受け取った SRM の管理下にあるストレージシステムを特定するために指定され，SRM は指定されたストレー

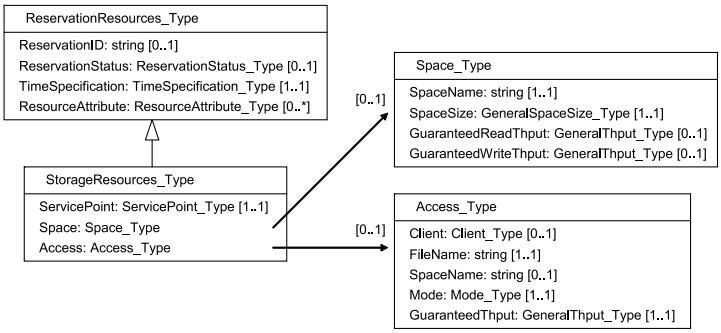

図 3　新規に定義したストレージ資源のデータ型
Fig. 3　Storage resources type.

```
bool create_bucket(char* path, off_t size,
              time_t start_time, time_t end_time,
              thput_t guaranteed_read_throughput,
              thput_t guaranteed_write_throughput)
bool create_bucket(char* path, off_t size,
              time_t start_time, time_t end_time,
              thput_t guaranteed_read_throughput,
              thput_t guaranteed_write_throughput,
              access_reserve_list_t& access_list,
              ticket_list_t& &ticket_list)
bool delete_bucket(char* path)
Object* create_object(char* path, Ticket& ticket,
                      OpenMode& mode)
Object* open_object(char* path,
                    access_type_t access_type,
                    Ticket& ticket, OpenMode& mode)
size_t write(char* buffer, size_t size)
size_t read(off_t offset, char* buffer, size_t size)
void close(Object* object)
```

図 4　主なクライアント API
Fig. 4　Major client APIs.

ジシステムに対して予約要求を転送する．性能[*1]とスペースを同時に予約する場合は，StorageResources_Type の配列を用意することになる．性能予約が成立した場合，提案ストレージシステムは SRM を介してクライアントに予約 ID を返す．クライアントはアクセス時にこの予約 ID を提案ストレージシステムに提示しなければならない．

2.3.2　クライアント API

提案ストレージシステムは予約とデータアクセスの両方を含むクライアント API ライブラリを提供する．**図 4** に主な API 関数を示す．*create_bucket()* と *delete_bucket()* はそれぞれスペースの予約とキャンセルのための関数であり，その他はデータアクセスに関する関数である．*create_object()* と *open_object()* では前記の予約 ID を *Ticket* として指定する必要がある．提案ストレージシステムではユーザ ID と予約 ID の組合せで予約が管理されているため，ユーザ認証の後にこの指定された予約 ID の照会により予約の正当性が確認される．Write のアクセスモードは create と append の 2 つだけであるため，*write()* では対象とする Object のオフセット位置を指定する必要がない．

[*1] GNS-WSI3 仕様の拡張データ型である StorageResources_Type においては，性能予約はアクセス予約と表現され，図 3 に示すように性能要求を記述するための Access_Type が用意されている．

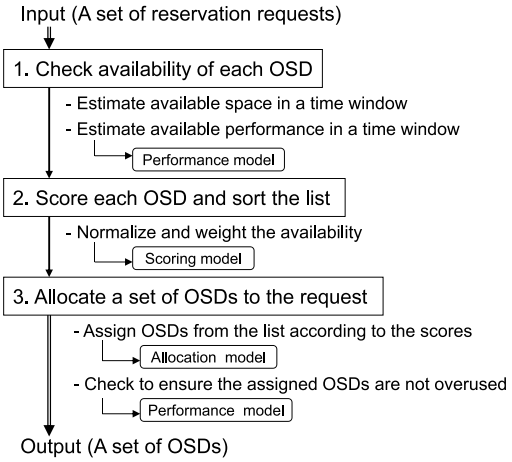

図 5　ストレージ資源の割当て手順
Fig. 5　Allocation flow.

2.3.3　ストレージ資源管理

提案ストレージシステムではディスク空間および各 OSD が提供できるスループットをストレージ資源とし，MGS が時間軸で空き状況と予約状況を管理する．そして，予約要求に対して**図 5** に示すステップに従って資源を割り当てる．

まず，MGS は予約要求を受け取ると，指定された時間枠における各 OSD の空き資源量のリストを作る．空き資源量は各デバイスが提供可能な最大能力から予約された分の能力を差し引くことで求まり，空きスペースの場合はディスクの最大容量から予約されたスペースの合計値を差し引けばよい．しかし，ディスクの空き性能（ここではスループット）の計算は，アクセスの混雑状況に応じたオーバヘッドを考慮しなければならず，何らかの性能予測モデルが必要になる．SSD を用いる場合には，たとえば，SSD 内部に実装されている処理アルゴリズムに基づく性能解析モデル [10] や機械学習を用いて実験結果から構築した性

表 1　開発したプロトタイプの制約
Table 1　Constraints of our prototype.

	Direct impact on users	Internal constraints
Design level	Read and write operations should not be mixed in a single access (from *open* to *close*).	
Implementation level (Not implemented yet)	I/O rate control is not supported for write access.	Write performance reservation against the OSD is exclusive to other performance reservations.

能モデル [11] をもとに性能予測を行うことが考えられる．そこで，提案ストレージシステムではストレージ管理者が任意の性能予測モデルを実装可能なように設計する．次にMGS は空き資源量に応じて OSD のスコアを決定し，スコア順にリストをソートする．そして，基本的には高いスコアが付けられた OSD から順に割当てを行う．このスコアの付け方と割当てアルゴリズムにもいくつかのモデルを考えることができ，ストレージ管理者が資源の割当てポリシとして任意のモデルを実装可能なように設計する．資源割当ての最終ステップでは，MGS は前記の性能予測モデルを再度用いて，新規の割当てが各 OSD の最大能力を超えていないことを確認する．これは性能予測モデルが線形的に計算されないことに起因する割当てミスを検知するために用意したステップである．

2.3.4　I/O 性能制御フレームワーク

性能保証を行うには，予約時間帯にディスク I/O スケジューリングやネットワークの帯域制御などの I/O 性能制御を有効にする必要がある．提案ストレージシステムでは，クライアントは必ず MGS に対してアクセスの開始要求を行い，その際に MGS が予約の有無を判断するため，MGS から SS あるいはその他のバックエンドの各機器に I/O 制御を指示する仕組みを用意する．そして，SS またはバックエンドの機器が指示に従って，組み込まれた I/O 制御機構を起動するようにする．具体的に SS と OSD の間では次のような設計にした．

SS は OSD に対する OPEN, CLOSE, READ, WRITE, REMOVE, GET_ATTR, および SET_ATTR を RPC インタフェースとしてクライアントに提供する．これに加えて，T-10 の OSD の標準規格に含まれる Capability モデル [12] を実装し，セキュリティ制御に加えて I/O 性能制御を行えるようにする．このモデルでは，MGS はクライアントからのアクセス要求の処理において，I/O 制御に関する命令とオブジェクトに対するパーミッションの両方を記した Capability を生成し，クライアントに返す．クライアントは SS に OPEN 要求を行う際に MGS から受け取った Capability を SS に提示する．SS は Capability を参照し，そこに記された命令に従って当該クライアントからの当該オブジェクトへのアクセスがある間，I/O 制御機構を動作させる．なお，MGS と SS の間では事前に安全な方法でキーを共有しておき，MGS がそのキーを用いて Capability に署名し，SS がそれを検証する仕組み [13] を導入することで不当な I/O 制御を防ぐことができる．

3.　提案ストレージシステムの実装

3.1　プロトタイプの概要

これまでに述べた設計に基づき，本研究では提案ストレージシステムを実現するためのストレージソフトウェアのプロトタイプを開発した．開発には C++を用い，Linux 上で動作確認を行った．また，単純な性能予測モデルを用いた貪欲法による資源割当てを実装し，予約に基づいて End-to-End で性能を確保するためにストレージネットワークとディスク I/O スケジューリングにおいて I/O 制御を行う仕組みをシステムに組み込んだ．MGS の実装には SQLite Version 3 を利用し，バケットとオブジェクトのメタ情報，予約情報を格納するようにした．SRM の開発には GNS-WSI3 仕様をサポートしている GridARS を用い，プロトタイプが提供する予約コマンドの実行を代行する Web Services に基づくストレージ資源予約インタフェースを実装した．

ただし，性能予約インタフェースは Read と Write の両方について実装したものの，Write の I/O 性能制御はまだ開発段階であるため，本プロトタイプではサポートしていない．また，OSD において Read と Write が競合したときの性能予測の方法も検討段階であり，ストレージシステム内部において MGS が行う OSD への Write 予約はその OSD に対する他の予約と排他的に行うように資源割当てに制約を加えた．すなわち，本プロトタイプには**表 1** にまとめたとおり，2.1 節で述べた提案ストレージシステムの設計上の制約と開発途上であるために限定的に実装されている機能が存在する．

次節以降では本プロトタイプにおける OSD，性能予測と資源割当ての手法，組み込んだ I/O 制御機構について詳細を述べる．

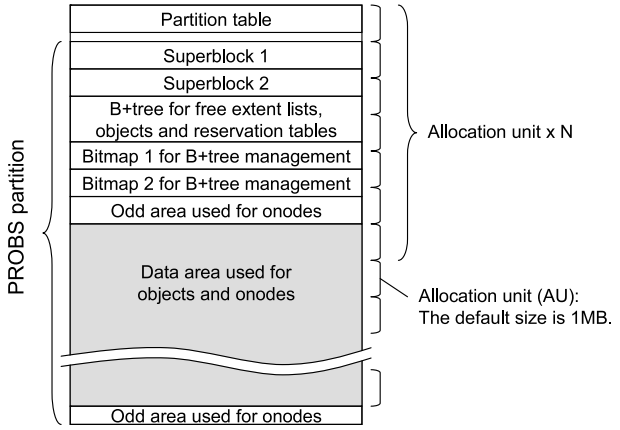

図 6　PROBS のデータ格納レイアウト
Fig. 6　Allocation layout of PROBS.

3.2　I/O 性能制御機構を備えたオブジェクトベース・ストレージデバイス

提案ストレージシステムを実現するにはストレージデバイスに直接アクセスし，スペース管理および I/O スケジューリングを制御できるオブジェクトベース・ストレージデバイス（OSD）が必要である．しかし，既存の OSD にはそうした機能を持つものはなく，本研究において新規に開発した．本論文ではこれは PROBS と呼ぶ．PROBS は Extent と B+tree を用いたオブジェクト・ファイルシステムである EBOFS [14] をもとに独自の変更を加えたものであり，スペース予約の機能を持つ．**図 6** に示すように，PROBS のパーティションはアドレスの先頭からスーパブロック，B+tree テーブル，B+tree のビットマップなどの格納領域があり，その後に Object や Onode（Object のメタ情報）が配置される．ただし，Object と Onode はフォーマット時に固定したオフセットを起点とし，AU（Allocation Unit）と呼ぶ割当て単位（デフォルトは 1 MB）を用いて割当てがなされる．その結果，PROBS のスペースは AU の数で管理でき，全 AU 数とある時間において予約された AU 数からスペースの空きを調べることができる．PROBS に対するスペース予約は MGS から SS を経由して行われる．MGS はスペース予約の際に得た ID を内部的に管理し，そのスペースに対する Write 要求が届いたときに 2.3.4 項で述べた Capability の中にその ID を埋め込む仕組みを実装した．

さらに，PROBS は Object へのアクセス開始時に目標スループットを与えると，それを満たすように I/O スケジューリングを行う機能を持つ [15]．SS は Capability に含まれる予約スループットを参照し，**図 7** に示すように PROBS に対してスループットの割当てを要求する．PROBS 内部では要求ごとに動的に FIFO キューが作成され，ディスクへの I/O を司る I/O スレッドが一定の I/O 間隔（スケジューリング・ラウンド）ごとに目標スループットを満たすように各キューの I/O 要求を処理する．これはトークンを用いた Weighted Round Robin（WRR）に基づいて実装している．また，本機能はバックエンドに SSD を用いることを前提としており，Read に関しては安定したスループットを保証可能である．Write に関しては，3.1 節で述べたように本プロトタイプではストレージシステム全体としては Write の I/O 性能制御をサポートしていないが，PROBS では基本機能はすでに実装されている．SSD 自体の Write 性能のばらつきが大きいという問題に対処するため，PROBS ではメタ情報の更新を抑制し，かつ定期的に書き込む工夫により，少ないオーバヘッドで性能保証を行えるようになっているが，さらなる改善が必要である．

3.3　資源割当て手法

資源割当て手法の実装は図 5 に示した性能予測モデル，スコアリングおよび割当てモデルのそれぞれについて行う必要がある．本プロトタイプでは，きわめて単純な性能予測モデルを用い，OSD が提供可能な最大スループットから予約済みのスループットを一定の割合のオーバヘッドを掛け合わせて差し引くこととした．ただし，表 1 に示すように Write アクセスを他の Write および Read アクセスと競合させないという制限を設けて，他のアクセスが予約されている場合，Write の空きスループットは必ず 0 となるようにし，Write アクセスが予約されている場合は Read，Write に関係なく空きスループットが 0 になるようにした．

なお，OSD が提供可能な最大スループットは PROBS の性能測定ツールを用いた予備実験により求めることにした．このツールは任意の数のスレッドを起動して，PROBS インタフェースを提供する OSD に同時にアクセスを行い，各スレッドの I/O スループットとそれらの合計スループットを測定する．ストレージシステムの管理者は最初に各 OSD に許す同時アクセス数の上限を決定し，その上限値以下の数のスレッドを起動して，ほぼ確実に I/O 制御が行える合計スループットの最大値を調べ，それを OSD の最大スループットとして本プロトタイプのパラメータに設定することになる．

スコアリングでは，スペースの空きよりもスループットの空きを重視するようにした．つまり，空きスループットが大きいほどスコアが高くなるようにし，空きスループットが同じ場合には，空きスペースが大きいほどスコアが高くなるようにする．そして，割当てモデルではできるだけ高いスコアの OSD から割り当てて，1 つのアクセスに対して割り当てる OSD 数（つまり，ストライプ数）を減らすようにした．図 5 のステップ 3 は利用可能な OSD が見つかるまで繰り返すこととし，性能要求を満たす最小のストライプ数からループを開始する．たとえば，200 [MB/sec] のスループットが要求され，各 OSD が提供できる最大スループットが 100 [MB/sec] の場合，ストライプ数を 2 に設定して割当てを試みる．しかし，別の予約により，100 [MB/sec]

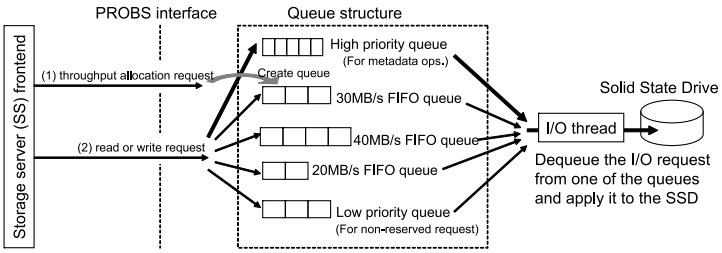

図 7　PROBS におけるスループット別に用意された I/O スケジューリング・キューの構成例
Fig. 7　An example of the I/O scheduling queues in PROBS.

の性能を提供できる OSD が 2 つも見つからない場合は，ストライプ数を 3 に増やして 67 [MB/sec] のスループットを提供できる OSD を 3 つ探索する．このループは必要な数の空き OSD が見つかるか，あらかじめ設定された最大ストライピング数に達するまで継続する．最大ストライピング数に達しても，必要な数の空き OSD が見つからない場合，予約要求は拒否されて，空き資源がない旨の通知がクライアントに返される．

3.4　クライアントとストレージサーバ間のネットワーク I/O 制御

クライアントとストレージサーバ（SS）間のネットワーク I/O を制御するため，プロトタイプシステムでは PSPacer [16] を組み込むこととした．PSPacer はイーサネット上のネットワークバンド幅を最大限に利用するために作られたソフトウェア・モジュールであるが，目標とするデータ転送レートを与えて，それに合わせた I/O 制御を行うことも可能である．3.1 節で述べたように本プロトタイプでは Write の I/O 性能制御はサポートしないため，Read アクセスについてのみ PSPacer を適用する．Read アクセスだけであるため，PSPacer に設定する目標レートは PROBS のディスク I/O スケジューリングと同様に，create あるいは open 要求時に Capability に含まれる情報の 1 つとして SS に伝える仕組みとした．SS は tc（traffic control）コマンドを内部的に実行し，IP アドレスとポート番号を指定して SS からクライアントへのデータ転送に対して目標レートを設定し，オブジェクトへのアクセスが終了すると設定を解除するように動作する．PSPacer を用いた理由は特別なハードウェアを要求しないという点と高い精度の制御が行えるという点であるが，類似の仕組みを Infiniband やファイバ・チャネルのネットワークのように QoS 機能を提供する商用のネットワークスイッチへ応用することも設計上可能である．

4.　評価実験

本章ではプロトタイプ・ソフトウェアを用いて提案ストレージシステムを構築し，予約操作のオーバヘッドと予約に基づく性能保証の効果を検証した実験の結果を示す．前者の実験では，コマンドラインと SRM インタフェースの両方において予約操作に要するコストを測定した．そして，後者の実験では構築したストレージシステムに対して同時に 2 つの Read アクセスを行い，性能が予約されている場合とそうでない場合を比較することで，予約と I/O 制御による効果を明らかにした．

4.1　予約操作に要する時間

本実験では MGS に対する予約操作の応答性能として，予約プロトコルのオーバヘッドを評価した．まず，SRM と MGS を同じマシン上（node-1）に配置し，同じマシン（node-1）または別のマシン（node-2）から予約要求を行って予約またはキャンセルが成立するまでの時間を測定した．実験には AMD Opteron 4Core 2376 2.3 GHz，8 GB のメモリを搭載したマシンを 2 台用い，MGS が使うバックエンドのディスクには OCZ Apex V3 を用いた．また，node-1 と node-2 は 1 Gbps のイーサネットで接続した．要求内容はスペースとそのスペースに対する Write 性能の同時予約またはそのキャンセルとし，Web Services に基づく SRM インタフェースを用いた場合とコマンドライン・インタフェースを用いて MGS に対して直接要求を行った場合の実行時間を測定した．

表 2 は a）SRM インタフェースを用いて node-2 上のクライアントから node-1 上の MGS にアクセスした場合（ただし，バックエンドにはダミーの MGS を利用し，SRM プロトコルの処理時間のみを測定），b）SRM インタフェースを用いて node-2 上のクライアントから node-1 上の MGS にアクセスした場合，c）コマンドライン・インタフェースを用いて node-1 上のクライアントから同じノード上の MGS にアクセスした場合，d）コマンドライン・インタフェースを用いて node-2 上のクライアントから node-1 上の MGS にアクセスした場合の 4 パターンを比較した結果である．SRM インタフェースでは *Initialize* 操作に一定の時間が必要であり，その後の *Reserve* または *Release* 操作には要求送信に要する時間（*Request*）とポーリングによる応答確認（*Confirm*），そして要求のコミット（*Commit*）の合計時間が必要である．また，本研究で開発した SRM のプロトタイプは，*Request* を受け取ると MGS のコマン

表 2　予約操作に要する時間
Table 2　Reservation cost.

a) SRM-client on node-2 connects to SRM/dummy-MGS-client on node-1.
b) SRM-client on node-2 connects to SRM/MGS-client on node-1.
c) MGS-client on node-1 locally connects to the MGS on node-1.
d) MGS-client on node-2 remotely connects to the MGS on node-1.

Operation		Execution time [msec]			
		a)	b)	c)	d)
Initialize		82.4		–	–
Reserve	Total	477	613	149	153
	Request	102	105	–	–
	Confirm	197	322	–	–
	Commit	177	185	–	–
Release	Total	386	511	137	141
	Request	107	108	–	
	Confirm	98	221	–	
	Commit	181	183	–	

ドライン・ツールを外部実行して，MGS に予約またはキャンセルを要求する．コマンドはノンブロッキングで実行され，*Confirm* によってその終了ステータスが確認される．なお，*Commit* は GNS-WSI3 仕様の二相コミットのための操作であり，本プロトタイプでは予約またはキャンセルが成立する場合は SRM サーバは MGS に対して何も行わず，SRM クライアントに ACK を返す．

表 2 の結果はコマンドライン・インタフェースに比べて SRM インタフェースの実行時間が大きいが，そのオーバヘッドの半分がポーリングを用いた *Confirm* と *Commit* を必要とする予約プロトコルが原因であることを示している．*Confirm* は予約ステータスが “confirmed” になるまで繰り返し行われる．本実験では *Confirm* のポーリング間隔を 100 [msec] に設定したため，1 回の *Reserve* 要求において 2〜3 回，*Release* 要求において 1〜2 回の *Confirm* が行われた．なお，本実験結果は現在時刻を起点とする予約，かつ予約してすぐにアクセスを開始するようなケースをサポートする場合に，アクセスを開始できるまでにどれくらいの遅延が生じるかを示す指標でもある．

次に MGS の性能のスケーラビリティを確認するため，コマンドライン・インタフェースを用いた上記の d）の実験と同様の予約操作を 10,000 回連続的に実行した．このとき，性能を予約する時間帯はそれぞれ異なるようにし，すべての予約要求が成功するようにした．実験の結果，予約に要する時間は線形的に増加し，最終的には 193 [msec] になった．

2.1 節に述べたように予約に基づく I/O が一定以上の時間続くことを想定すれば，本実験で示された予約操作に要する時間は両インタフェースとも十分に小さく，多数の予約が入っていても処理の大幅な低下は見られない結果となった．一方，本実験は予約プロトコルのオーバヘッドに着目したため，用いた予約内容はそれほど複雑でなく，MGS に実装した割当て手法も単純なものである．資源が競合して予約が失敗したり，複数のクライアントから同時に予約が行われたりするテストケースでは予約プロトコルのオーバヘッドよりも資源探索のオーバヘッドが大きい可能性がある．また，MGS は予約操作だけでなく，open や close などのメタ情報に関する操作も処理しなければならない．Read や Write は MGS を介さずに行われるため，MGS が高負荷であっても影響はないが，open や close において遅延が発生する可能性がある．そうした資源探索も含めた予約操作のオーバヘッド，予約とメタ情報の操作性能を合わせた MGS の処理能力の評価は今後の課題である．

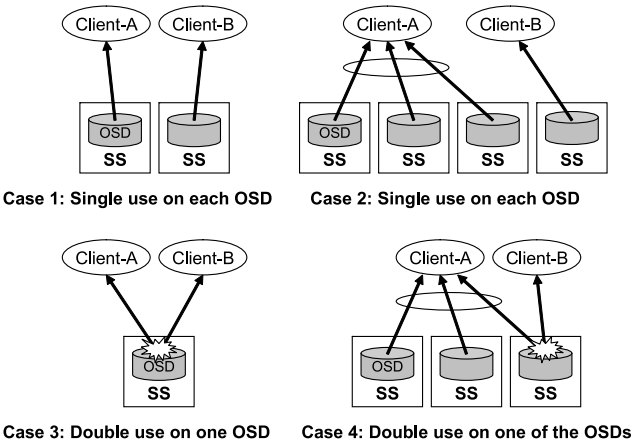

図 8　同時アクセス時の性能計測における実験パターン
Fig. 8　Experimental cases.

4.2　予約と I/O 性能制御の効果

予約および I/O 性能制御の効果を検証するため，プロトタイプ・ソフトウェアを用いて構築したストレージシステムに対して同時に Read アクセスを行ったときの各アクセスのスループットを測定した．具体的には図 8 に示すように Client-A と Client-B が同時に Read を実行する 4 つの実験パターンを用意した．そして，OSD へのアクセスが競合する Case 3 と 4 においてはプロトタイプの設定を切り替えて，PSPacer および PROBS による I/O 性能制御を行う場合と行わない場合を試した．なお，Case 4 は Client-A が 3 つの OSD に対して並列にアクセスを行い，Client-B がそのうちの 1 つの OSD にアクセスする状況を示している．各実験では Client-B を先に起動し，3 秒後に Client-A を起動した．そして，Client-A の実行が終了してから Client-B が停止するようにし，Client-A はつねに Client-B の影響を受けるという状況を用意した．

実験環境としては AMD Opteron 8Core 6128 2.0 GHz，8 GB のメモリを搭載したマシンをクライアント，AMD Opeteron 4Core 2376 2.3 GHz，8 GB のメモリを搭載したマシンを SS として用い，全マシンを 10 Gbps のイーサネットで接続した．MGS は Client-A を実行するマシン上に

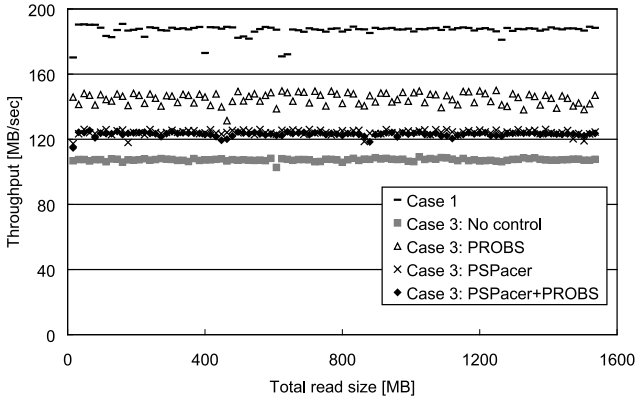

図 9　Case 1 と 3 における Client-A のスループット（要求：120 [MB/sec]）

Fig. 9　Measured throughputs of Client-A in Cases 1 and 3 (Request: 120 [MB/sec]).

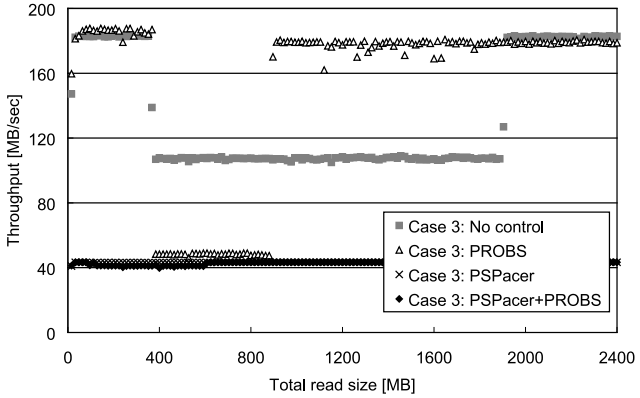

図 10　Case 3 における Client-B のスループット（要求：40 [MB/sec]）

Fig. 10　Measured throughputs of Client-B in Cases 3 (Request: 40 [MB/sec]).

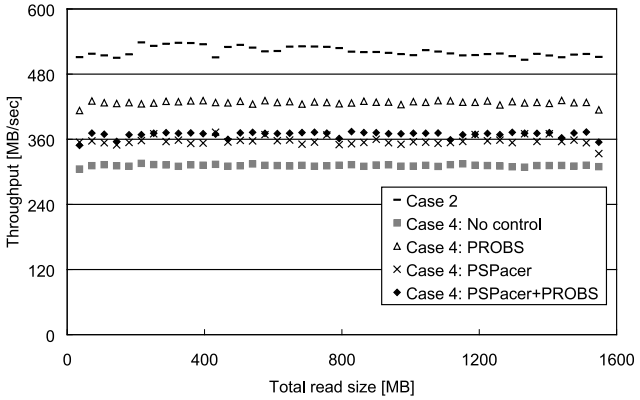

図 11　Case 2 と 4 における Client-A のスループット（要求：360 [MB/sec]）

Fig. 11　Measured throughputs of Client-A in Cases 2 and 4 (Request: 360 [MB/sec]).

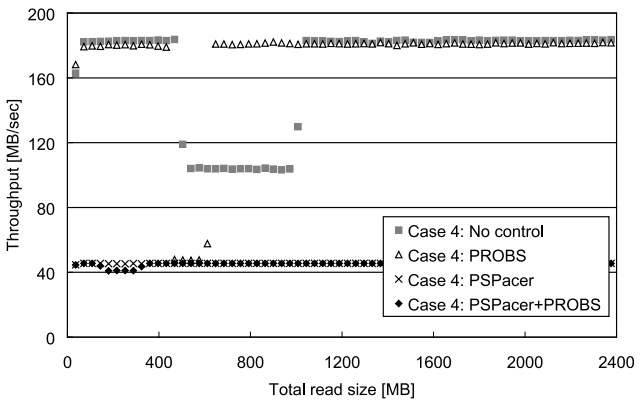

図 12　Case 4 における Client-B のスループット（要求：40 [MB/sec]）

Fig. 12　Measured throughputs of Client-B in Cases 4 (Request: 40 [MB/sec]).

起動した．そして，SS では OCZ Vertex 120 G を PROBS のバックエンドのストレージデバイスとして用い，3.3 節で述べた予備実験に基づいて OSD の最大スループットを 192 [MB/sec] に設定した．この値は同時に 2 つの Read アクセスを実行し，それぞれに対して 99.8%の確率で要求スループットを提供できたときの各要求スループットの最大合計値である．また，ストレージネットワークは 10 Gbps であることから，アクセスが競合した際は OSD がボトルネックになる環境である．

Case 1 と 3 の実験では，Client-A と Client-B の要求スループットはそれぞれ 120 [MB/sec] と 40 [MB/sec]，クライアントプログラムにおける Read の I/O サイズはいずれも 16 MB に設定した．Case 2 と 4 の実験では Client-A の要求スループットを 360 [MB/sec]，実行時のストライプサイズを 1 MB に設定した．他方，Client-B の要求スループットは 40 [MB/sec] に設定した．Read の I/O サイズはいずれも 36 MB に設定した．

以降の実験結果は予約の有無と I/O 性能制御の有無に基づくシナリオに沿って説明する．

4.2.1　占有戦略に基づく資源予約の効果

まず，性能予約が行われた場合に，ストレージシステムが異なる OSD をそれぞれのアクセスに対して割り当てる占有戦略をとるシナリオを考える．この場合，性能が予約されていれば図 8 に示す実験パターンの Case 1 と 2 に相当する割当てが行われる．しかし，性能予約がなされていない場合は，OSD へのアクセスが競合する可能性があり，たとえば図 8 の Case 3 と 4 に相当する状況が発生する．つまり，性能予約がある場合に提案ストレージシステムが Client-A に対して提供できるスループットは**図 9** の Case 1 および**図 11** の Case 2 である．それに対して，性能予約がなくアクセスが競合した場合の Client-A のスループットは図 9 の Case 3：No control および図 11 の Case 4：No control である[*2]．このとき，Client-B が享受したスループットは**図 10** の Case 3：No control および**図 12**

[*2] 開発したプロトタイプは実際には予約のないアクセスを受け付けないため，これは評価実験のために作り出した特殊な状況である．

の Case 4：No control である．

図 9 と図 11 より，アクセスが競合していなければ Client-A は要求値以上の性能を得ることができるが，アクセスが競合した場合には要求値以下となることが分かる．特に Case 4：No control ではアクセスが競合する OSD がボトルネックになり，Client-A は 309 [MB/sec] のスループットしか得ることができていない．アクセスが競合した際の Client-B のスループットは図 10 と図 12 の No control が示すように 103〜109 [MB/sec] であり，Client-A のスループットが Client-B の影響により低下していると考えることができる．また，図 9 より Client-A が要求するスループットが 106 [MB/sec] 以上であるなら，性能予約を行って Case 3 ではなく Case 1 の状況を作る必要があることが分かる．同様に図 11 より，309 [MB/sec] 以上のスループットが必要であるなら性能予約を行って Case 2 の状況を作る必要があることが分かる．

4.2.2　I/O 性能制御をともなう資源予約の効果

次に，性能予約が行われた場合に，ストレージシステムは OSD やストレージネットワークの性能に余裕があれば，Case 3 と 4 のように OSD への同時アクセスを許し，I/O 性能制御機構を用いて個々のアクセスのスループットを制御するシナリオを考える[*3]．図 9〜図 12 において PSPacer は PSPacer のみ，PROBS は PROBS のみ，PSPacer+PROBS は両方の I/O 性能制御を有効にした場合の結果である．性能が予約されていない場合はすべての I/O 性能制御を無効にした場合と考えることができ，その結果はこれらの図の No control に相当する．また，PSPacer で指定する目標帯域は，クライアントプログラムが要求するデータ（ペイロード）だけでなく，ストレージシステムのプロトコルにおけるヘッダを含む全転送量を考慮した値である必要がある．そこで予備実験を行い，Case 1 と 3 の実験においては要求帯域の 15%，Case 2 と 4 の実験においてはストライピング処理のオーバヘッドも加味して，要求帯域の 20%を見込んだ値を指定するようにした[*4]．PROBS では 8 MB ごとにそれぞれのアクセスの I/O 要求が適切な割合で処理されるようにスケジューリング・パラメータを設定した．

図 9 と図 11 より，PSPacer と PROBS を有効にすることで，Client-B のアクセスが同時に存在していても Client-A の要求スループットを満足できることが分かる．そして図 10 と図 12 より，アクセスが競合している間，Client-B のスループットは 40 [MB/sec] の要求スループットを下回らない程度に抑制されることが分かる．ただし，PROBS だけで制御する場合は Client-A のアクセスがないときには Client-B は 180 [MB/sec] のスループットを得るのに対し，PSPacer による制御を有効にした場合は，Client-B のスループットはつねに 40 [MB/sec] 付近に抑制される．なお，今回の実験では Case 3：PSPacer，Case 3：PROBS，Case 4：PROBS のようにいずれか 1 つの I/O 性能制御だけでも要求スループットを満たすことができた．しかし，Case 4：PSPacer では Client-A の要求スループットを若干であるが，満たせていない．また，ストレージネットワークのスイッチの許容帯域が小さい場合には PROBS だけでは制御が困難であり，将来的に Write の I/O 性能制御をサポートする場合には PSPacer だけでは制御が不可能であり，両方の I/O 性能制御が必要になる．

一方，I/O 性能制御を行うことによる PROBS と PSPacer のオーバヘッドは次のとおりである．別に行った実験により，SSD から直接，1 MB 単位でデータを読み出す際のスループットが 99.5%の確率で 235 [MB/sec] 以上であるのに対し，I/O 性能制御を無効にし，かつ 2 スレッドが同時に Read を実行した場合の PROBS の合計スループットは 99.8%の確率で 211 [MB/sec] 以上であることが分かっている．そして，I/O 性能制御を有効にした PROBS では，先述のように要求スループットの合計が 192 [MB/sec] 以下であれば 99.8%の制御が可能である．つまり，本実験環境では I/O 性能制御を除く PROBS のオーバヘッドは 10.2%であり，I/O 性能制御のオーバヘッドは 9%である．PSPacer については帯域を制限するための処理により CPU に負荷がかかる．それは同時に制御する接続数が 1 から 160 に増えた場合に 4%ほど CPU 利用率を上昇させる程度であり [17]，本実験では CPU の性能に十分余裕があった．ただし，提案システムでは先述のようにクライアントが要求するペイロードだけでなく，転送量全体を考えたスループットを PSPacer の目標帯域に指定する必要があり，今回の実験環境ではクライアントの要求スループットの 15〜20%をそのオーバヘッドとして考慮する必要があった．

5.　応用例

本提案ストレージシステムはクラウドやグリッドにおいて高スループットのストレージアクセスを要求するアプリケーションへの応用を想定している．すなわち，そのようなアプリケーションでは 2.1 節で述べた利用上の制約である，1）予約が必須であること，2）アクセスパターンが既知であり，Read と Write の混在のない逐次的なアクセスであることが問題でないか，問題があったとしても確実に性能を得られることの方が重要である場合が少なくないと考えている．以下では考えられるいくつかの応用例を紹介する．

クラウド環境の運用においてはデータセンタ内にある多数のサーバノードに対して OS のイメージ，仮想化された実行環境のイメージ，アプリケーション・データなどを展

[*3] I/O 性能制御を有効にした場合の評価のため，ここでは Case 1 と 2 の状況は考えない．
[*4] 開発したプロトタイプでは，このオーバヘッドを設定ファイルにて指定できるようになっている．

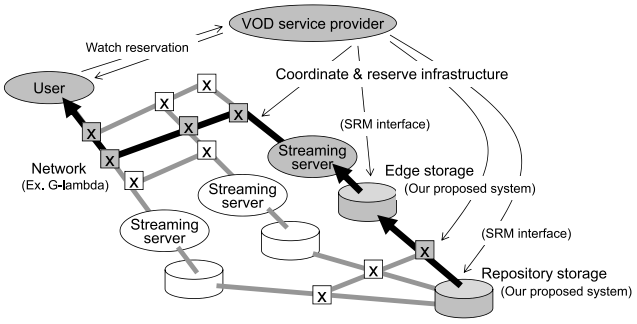

図 13　提案ストレージシステムと光パスのネットワーク帯域予約を用いた映像配信への応用例

Fig. 13　An example use of a streaming service using performance assured storage and network based on the reservations.

開したり，実行後に変更されたイメージをストレージに退避したりする作業の頻度が高く，いずれも非常に高スループットのI/Oを要求する．これらの実行はあらかじめ開始時刻を決めて実施できる場合があり，逐次的なデータアクセスに限定可能であることから提案ストレージシステムとの親和性が高い．また，性能や機能に応じて課金を行う場合には，予約に基づく性能保証の仕組みとの連携を考えることもできる．従来的なデータセンタ運用においても定期的にデータのフルバックアップを行う用途では性能予約が可能である．提案ストレージシステムによって性能が保証されることで，効率的かつ安全なフルバックアップ作業が可能になる．

グリッド環境を利用する科学アプリケーションには大規模な実験や高精度の観測で得られた大容量のデータを扱うものが多い．その中でも，観測したペタバイト規模のデータを広域に存在する高性能計算機を用いて解析するシナリオ [18] では，ネットワークの帯域予約と合わせて両端のストレージシステムのスループットを予約してデータ転送を行う応用が考えられる．さらに，データの転送先において，ストレージシステムに保存された大規模データをいっせいに読み込んで大規模な並列プログラムを実行するという応用も考えられ，提案ストレージを適用できる可能性が高い．

これらのうち，本研究では高精細映像を多数のユーザに配信するビデオ・オンデマンドの配信システムに提案ストレージシステムを適用した．これは配信に利用するネットワークやストレージなどの資源を予約に基づいて管理する仕組みを実装したものであり，**図 13** に示すように，まず視聴者はWeb上の受付サイトから映像コンテンツと視聴時間を予約する．そして，映像配信サービス提供者が視聴開始に間に合うようにコンテンツが格納されたリポジトリのストレージから配信サーバのストレージまでコンテンツを事前に転送する．このとき，両ストレージのスループットとネットワーク帯域が同時に予約される．さらに，実際の配信のために配信サーバからユーザのPCまでのネットワーク帯域と，配信サーバのバックエンドにあるストレージに対するI/Oスループットを別に予約する．本研究ではGNS-WSI3仕様に基づくインタフェースを有するSRMを開発することで，提案ストレージシステムと光パス・ネットワークを同時に予約することに成功し，ハイビジョン映像の配信を行うデモ実験を実施して図13の配信システムが実現可能であることを示した [19]．

6. 関連研究

Open Grid Forum（OGF）[20] では別のSRM（Storage Resource Management）インタフェースが標準仕様として提案されている [21] [*5]．OGF-SRMはディスクアレイやテープシステム，並列ファイルシステムなどの様々なバックエンドからなるグリッド上のストレージサービスを同じインタフェースで使えるようにするために決められた標準仕様であり，Web Servicesに基づいたインタフェースを提供する．たとえばLHC（Large Hadron Collider）[22] の実験で生成されたデータを各地の計算機に分散して解析するようなシナリオでは大規模なデータ転送が必要であるので，転送先のディスクスペースを確保するためにスペース予約が可能になっている．また，スペースの予約時にデータ損失の可能性やアクセス遅延のポリシを合わせて指定することが可能である．それに対して，本論文で述べたSRMはOGF-SRMと類似の予約概念を持つものの，アクセス時間を指定してアクセス時のスループットをユーザが明示的に予約できる機能を特徴としている．

Chuangらの論文 [23] は予約に基づいてコンテンツの複製を行うことでQoSを保証する分散ストレージサービスを提案している．ベストエフォートでキャッシュを保持するのとは異なり，予約されたアクセスのQoS要件を満たすように最適な複製の数や配置が決められる．ただし，Writeアクセスの QoS は考慮されていない．

Hippodrome [24] と MINERVA [25] はI/O負荷とそれに対するストレージシステムの性能を予測し，システムの効率的な利用と性能要件の充足の両方を満たす設定を自動的に行う機能を提供する．I/Oの要求間隔，要求サイズ，ディスク上のキュー長とアクセスの連続性を考慮した性能モデルを用いて負荷解析が行われ，最適な設定（資源割当て，デバイスの設定，RAIDパラメータなど）が見つけられる．SMART [26] は同様の機能を提供し，かつ時間枠を決めて負荷解析を行い，資源の割当てなどを行う．本研究で開発したプロトタイプは単純な性能予測モデルと資源割当て手法しか実装していないが，このような既存の性能予測や資源の割当て手法は提案ストレージシステムに適用可

[*5] 本論文では表記を簡易化するため，本研究で設計したSRMを単にSRMと記し，OGFで提案されているSRMをOGF-SRMと記す．ただし，OGF-SRMは本論文のSRM以前に提案されている．

能であり，より効率的な資源の割当てを可能にするものである．

映像配信の分野では QoS を提供するための研究が数多くなされている．それらの初期の研究の 1 つである Tiger Shark [4] では，連続したファイルアクセスにおいて内部的に I/O 資源（ディスクバンド幅，スイッチ性能，バッファ空間など）を予約する．新しいアクセス要求が到着すると，Tiger Shark は既存のクライアントに対して十分なディスク・スループットが得られることを確認してからサービスを開始する．サービス中のアクセスに対してはデッドライン・スケジューリングを適用し，要求された速度でデータの Read を行えるようにする．ストリーミングの QoS 要件を満足させるためのディスク I/O スケジューリングは，この分野において最も活発に行われてきた研究の 1 つである [5], [6]．また，本研究で開発した PROBS のもとになっている EBOFS に関するプロジェクトでは，AQuA [27] と Q-EBOFS [28] の研究がある．Q-EBOFS は I/O 性能別のキューイングに加え，バッファキャッシュの管理を行うことでアクセス間の性能分離を実現しており，類似の手法が PROBS にも応用されている．一方，Façade [29] はアプリケーションとストレージデバイスとの間のレイヤにおいてアプリケーションの I/O 要求をすべて捕捉して性能をモニタリングし，その結果を I/O 制御に反映させる仕組みを用いており，精度の高い性能保証を実現している．

これらの I/O 性能制御の研究はアクセス要求や I/O 負荷に応じて受動的に制御を開始するアプローチをとっており，それに対して本論文では予約に基づいて制御を行うアプローチを提案している．もちろん，より細粒度で I/O 性能を制御できれば，より精度の高い性能保証が可能になり，提案ストレージシステムの応用対象も広がる．すなわち，PROBS や PSPacer に限らず，その他の優れた I/O 性能手法を組み込むことで提案ストレージシステムの性能保証のレベルを改善可能である．また，SSD などのフラッシュを用いたストレージデバイスは Seek に要する時間が一定かつほぼ 0 であり，PROBS で一部実装したように，そうした特徴を活用してより精度の高い I/O 性能制御を実現することも可能であろう．

7. まとめと今後の課題

本論文では，ユーザによる明示的な予約に基づいて性能保証を行うアプローチを提案し，クラウドやグリッドにおいて用いられることを想定した分散ストレージシステムにおいて，そのアプローチを実現するアーキテクチャを提案した．その提案アーキテクチャに沿って，I/O 性能制御とスペース予約が可能なオブジェクトベース・ストレージデバイスである PROBS を開発し，ストレージネットワークの帯域制御が可能な PSPacer と合わせてプロトタイプに組み込み，ストレージデバイスからクライアントまでの Read アクセスの I/O パスにおいてスループットを確保する仕組みを実装した．この End-to-End の I/O 性能制御を予約に基づいて行うことで，ストレージシステムに対して同時に複数のアクセスが行われた場合でも，ストレージデバイスの利用を排他制御する対処だけでなく，性能に余裕があればストレージデバイスへの同時アクセスを許し，性能保証が行えることを示した．さらに，GNS-WSI3 仕様の資源定義をストレージ資源に拡張した予約インタフェースを提案し，提案ストレージシステムへの予約を代行する SRM を開発した．SRM により提案ストレージシステムとその他の IT 資源の同時予約が可能になることと，ストレージのスループットとネットワークの帯域の同時予約に基づいて映像配信を実際に行ったデモおよびその他の応用例を提示した．このように，アプリケーションの実行時にストレージ資源の不足が判明して性能が得られないという問題に対して，予約に基づいて性能保証を行う提案アプローチが有用であることを明らかにした．

本研究の今後の課題としては以下があげられる．

- Write I/O 性能制御の実装と評価：本論文では Read アクセスについてのみ I/O 性能制御を実装して評価を行った．しかし，Write アクセスについても同様の仕組みで I/O 性能制御を実現できる見込みである．また，開発したプロトタイプでは，ストレージデバイスを占有できたとしても I/O パス上の別のデバイス（クライアントノード，ネットワークスイッチなど）が他の Write アクセスと競合する場合が考えられ，Write の I/O 性能制御の実装は重要な課題である．そこで，Write の I/O 性能制御に PSPacer を組み込み，PROBS と組み合わせて利用したときの評価・改善を行う予定である．そのうえで，OSD において Read と Write が混在した場合の性能予測手法の検討も進めていきたい．
- 性能保証レベルの向上：PROBS の Write の I/O 性能制御は基本的な実装を完了しているが，SSD では内部実装されているウェアレベリングやガベージコレクションにより，Write の I/O 性能が Read より格段に不安定であるという問題が残っている．SSD の性能モデリングに関する研究 [10], [11] や 6 章で紹介した既存手法を取り入れるなどして I/O 性能制御の精度を高める必要がある．
- 機能改善：開発したプロトタイプでは OSD の最大能力や PSPacer のオーバヘッドは予備実験により求める必要がある．しかし，提案ストレージシステムの実用化を考えた場合，これらをシステムが自動的に取得する仕組みが必要である．さらに，OSD の最大能力はディスクの使用量，長期間の利用や障害など様々な理由により変化する可能性があり，定期的な性能調査を行うようにすべきである．一方，予約機能の使い勝手の改善も課題である．オブジェクトへのアクセス開始

時に予約 ID を渡す操作を隠蔽または洗練された手順にすること，予約に基づいたアクセスと予約を行っていないベストエフォートのアクセスの共存を可能にすること，予約後すぐに利用を開始するような予約のサポート，性能予約と連携したオブジェクトの複製機能などが考えられる．

- 実際的な評価：より実際的なアプリケーションにおいて予約や予約に基づく I/O 性能制御の有効性を検証することも重要な課題である．特にメタ情報や予約の操作が集中した場合の MGS の性能，予約や性能制御が及ぼすストレージシステム全体の利用効率への影響などを調査する必要がある．

謝辞　本研究の一部は文部科学省イノベーションシステム整備事業「光ネットワーク超低エネルギー化技術拠点」，および日本学術振興会科学研究費補助金（23680004）の助成によるものである．

本研究を行うにあたり，PSPacer の利用に関して多くのご支援をいただいた高野了成，岡崎史裕（産総研）の両氏に深く感謝いたします．

参考文献

[1] Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., Miyamoto, T., Otani, T., Tanaka, H., Suzuki, M., Sameshima, Y., Imajuku, W., Jinno, M., Takigawa, Y., Okamoto, S., Tanaka, Y. and Sekiguchi, S.: G-lambda: Coordination of a Grid scheduler and lambda path service over GMPLS, *Future Generation Computing Systems*, Vol.22, pp.868–875 (2006).

[2] G-lambda Project, available from ⟨http://www.glambda.net/⟩.

[3] Bigelow, D.O., Iyer, S., Kaldewey, T., Pineiro, R.C., Povzner, A., Brandt, S.A., Golding, R.A., Wong, T.M. and Maltzahn, C.: End-to-end Performance Management for Scalable Distributed Storage, *Proc. 2nd International Workshop on Petascale Data Storage* (*PDSW'07*), pp.30–34 (2007).

[4] Haskin, R.L. and Schmuck, F.B.: The Tiger Shark File System, *Proc. 41st IEEE International Computer Conference* (1996).

[5] Reddy, A.L.N. and Wyllie, J.: Disk Scheduling in a Multimedia I/O System, *Proc. 1st ACM International Conference on Multimedia*, pp.225–233 (1993).

[6] Shenoy, P.J., Goyal, P., Rao, S. and Vin, H.M.: Symphony: An Integrated Multimedia File System, Technical Report CS-TR-97-09, University of Texas, Austin (1998).

[7] Amazon S3, available from ⟨http://aws.amazon.com/s3/⟩.

[8] GNS-WSI version 3 (Grid Network Service – Web Services Interface, version 3), available from ⟨http://www.g-lambda.net/wordpress/wp-content/uploads/2008/11/g-lambda-gns-wsi3.pdf⟩.

[9] Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: GridARS: An Advance Reservation-Based Grid Co-allocation Framework for Distributed Computing and Network Resources, *Proc. 13th Workshop on Job Scheduling Strategies for Parallel Processing*, LNCS 4942, pp.152–168 (2007).

[10] Boboila, S. and Desnoyers, P.: Performance Models of Flash-based Solid-State Drives for Real Workloads, *Proc. 27th IEEE Symposium on Mass Storage Systems and Technologies* (*MSST*), pp.1–6 (2011).

[11] Li, S. and Huang, H.H.: Black-Box Performance Modeling for Solid-State Drives, *Proc. IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems* (*MASCOTS*), pp.391–393 (2010).

[12] T10: SCSI Object-Based Storage Device Commands-2 (OSD-2) Working Draft, available from ⟨http://www.t10.org/⟩.

[13] Factor, M., Nagle, D., Naor, D., Riedel, E. and Satran, J.: The OSD Security Protocol, *Proc. 3rd IEEE International Security in Storage Workshop*, pp.29–39 (2005).

[14] Weil, S.A.: Leveraging Intra-object Locality with EBOFS, Technical Report UCSC CMPS-290S (2004).

[15] 谷村勇輔，鯉江英隆，工藤知宏，小島　功：SSD を用いたオブジェクトベース・ストレージデバイスの I/O 性能制御，情報処理学会研究報告，Vol.2011-HPC-130, No.31, pp.1–8 (2011).

[16] Takano, R., Kudoh, T., Kodama, Y., Matsuda, M., Tezuka, H. and Ishikawa, Y.: Design and Evaluation of Precise Software Pacing Mechanism for Fast Long-Distance Networks, *Proc. 3rd International Workshop for Fast Long Distance Networks* (2005).

[17] Takano, R., Kudoh, T., Kodama, Y. and Okazaki, F.: High-resolution Timer-based Packet Pacing Mechanism on the Linux Operating System, *IEICE Trans. Communications*, Vol.E94.B, No.8, pp.2199–2207 (2011).

[18] The Worldwide LHC Computing Grid (WLCG) project, available from ⟨http://lcg.web.cern.ch/lcg/public/⟩.

[19] Yamada, K., Tsukishima, Y., Matsuda, K., Jinno, M., Tanimura, Y., Kudoh, T., Takefusa, A., Takano, R. and Shimizu, T.: Joint Storage-Network Resource Management for Super High-Definition Video Delivery Service, *Proc. OFC/NFOEC* (*National Fiber Optic Engineers Conference*), *OSA Technical Digest* (*CD*), Optical Society of America (2001).

[20] Open Grid Forum, available from ⟨http://www.ogf.org/⟩.

[21] Sim, A., Shoshani, A., Badino, P., Barring, O., Baud, J.-P., Corso, E., Witt, S.D., Donno, F., Gu, J., Haddox-Schatz, M., Hess, B., Jensen, J., Kowalski, A., Litmaath, M., Magnoni, L., Perelmutov, T., Petravick, D. and Watson, C.: GFD-R-P.129: The Storage Resource Manager Interface Specification Version 2.2, Open Grid Forum (2008).

[22] LHC – The Large Hadron Collider, available from ⟨http://lhc.web.cern.ch/⟩.

[23] Chuang, J.C.-I. and Sirbu, M.A.: Distributed Network Storage Service with Quality-of-Service Guarantees, *Proc. Internet Society INET'99 Conference* (1999).

[24] Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M. and Veitch, A.: Hippodrome: Running circles around storage administration, *Proc. 1st USENIX Conference on File and Storage Technologies* (2002).

[25] Alvarez, G.A., Borowsky, E., Go, S., Romer, T.H., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A. and Wilkes, J.: MINERVA: An Automated Resource Provisioning Tool for Large-Scale Storage Systems, *ACM Trans. Computer Systems*, Vol.19, No.4, pp.483–518 (2001).

[26] Yin, L., Uttamchandani, S., Korupolu, M., Voruganti, K. and Katz, R.: SMART: An Integrated Multi-Action Advisor for Storage Systems, *Proc. USENIX Annual Technical Conference* (2006).
[27] Wu, J. and Brandt, S.A.: The Design and Implementation of AQuA: An Adaptive Quality of Service Aware Object-Based Storage Device, *Proc. 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies*, pp.209–218 (2006).
[28] Wu, J.C. and Brandt, S.A.: Providing Quality of Service Support in Object-Based File System, *Proc. 24th IEEE Conference on Mass Storage Systems and Technologies*, pp.157–170 (2007).
[29] Lumb, C.R., Merchant, A. and Alvarez, G.A.: Facade: Virtual Storage Devices with Performance Guarantees, *Proc. 2nd USENIX Conference on File and Storage Technologies*, pp.131–144 (2003).
[30] Tanimura, Y., Koie, H., Kudoh, T., Kojima, I. and Tanaka, Y.: A Distributed Storage System Allowing Application Users to Reserve I/O Performance in Advance for Achieving SLA, *Proc. 11th ACM/IEEE International Conference on Grid Computing*, pp.193–200 (2010).

谷村 勇輔　（正会員）

昭和 51 年生．平成 16 年同志社大学大学院工学研究科知識工学専攻博士課程修了．同年独立行政法人産業技術総合研究所グリッド研究センター特別研究員．平成 17 年同センター研究員．平成 20 年より同所情報技術研究部門研究員．博士（工学）．最適化手法等の並列応用計算，グリッドにおけるプログラミングミドルウェア，グリッドやクラウド向けの並列分散ストレージ，並列データ処理に関する研究に従事．HPCS2005 論文賞，HPC ASIA 2009 Best Paper Award 受賞．

鯉江 英隆

昭和 47 年生．平成 8 年静岡大学大学院理学研究科物理学専攻修士課程修了．同年（株）数理技研入社．クラスタ，グリッドに関するシステム開発に従事．

工藤 知宏　（正会員）

平成 3 年慶應義塾大学大学院理工学研究科博士課程単位取得退学．東京工科大学助手，講師，助教授を経て，平成 9 年新情報処理開発機構並列分散システムアーキテクチャつくば研究室長，平成 14 年より産業技術総合研究所，現在，同所情報技術研究部門副研究部門長．博士（工学）．並列分散処理，通信アーキテクチャに関する研究に従事．電子情報通信学会，IEEE-CS 各会員．

小島 功　（正会員）

昭和 59 年京都大学大学院工学研究科情報工学専攻修了，同年通産省電子技術総合研究所入所．現在，独立行政法人産業技術総合研究所情報技術研究部門サービスウェア研究グループ長．分散・並列データベースや e-サイエンス基盤に関する研究に従事．ACM 会員．Open Grid Forum データベース WG 共同議長．OGC（Open Geospatial Consortium）メンバ．

田中 良夫　（正会員）

昭和 40 年生．平成 7 年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．平成 13 年 4 月より独立行政法人産業技術総合研究所．現在，同所情報技術研究部門主幹研究員．博士（工学）．分散環境における高性能計算，ミドルウェアおよびセキュリティに関する研究に従事．平成 18 年度情報処理学会論文賞．平成 21 年度科学技術分野の文部科学大臣表彰科学技術賞（研究部門）．ACM，IEEE 各会員．