# 高速ネットワークを流れるパケットデータの蓄積・活用に向けた分散オブジェクトストレージの提案

田村 雅寿　飯澤 健　前田 宗則　加藤 純　熊野 達夫　野村 祐士　小沢 年弘

(株)富士通研究所 〒211-8588 川崎市中原区上小田中 4-1-1

E-mail: {masahisa, iizawa.ken, maeda.munenori, jun.kato, kumano_tatsuo, nomura.yuji, t.ozawa}@jp.fujitsu.com

**あらまし**　近年、ネットワークを流れるデータ量の増大とともに、複雑なトラフィックパターンのアプリケーションも増加してきており、通信品質やトラブルシューティング等に向けたパケットデータ解析の重要性が高まってきている。特に、ネットワークや端末にどのような現象が発生したかを詳細に診断するために、統計情報に基づく解析だけでなく、個々のパケットデータを蓄積・解析することへの要望が強まっている。本論文では、パケットデータの高スループットな蓄積と、セッションの高速な検索を両立させる分散オブジェクトストレージを提案する。パケットデータのような小さいサイズのデータを、古いデータの削除と検索を行いつつも高スループットに蓄積できる方式の提案・評価を行い、その評価結果を示す。

**キーワード**　パケットキャプチャ, セッション検索, 分散オブジェクトストレージ

## 1. はじめに

近年、ネットワークを流れるデータ量の増大とともに、ネットワーク設計時には想定していなかった複雑なトラフィックパターンによるトラブルも発生するようになってきており、それらの解決に向けたパケットデータ解析の重要性が高まってきている。なかでも、ネットワークや端末にどのような現象が発生したかを詳細に診断するために、統計情報やサンプリングデータに基づく解析だけでなく、個々のパケットデータを全て蓄積・解析する要望が強まっている。また、一部の標的型攻撃では、長期間に渡って巧妙に少しずつ準備がなされていくことが判明している。そういった長期間にわたるトラフィックパターンも解析可能なように、長期間にわたるデータの蓄積も求められるようになってきている。

セキュリティや通信品質の解析のためには、大量のパケットデータの蓄積と並行して解析を行う必要がある。解析には、パケットデータ群に含まれているセッションや通話を切り出して処理されることが多く、必要とするセッションや通話を検索する機能にも高い性能が要求される。高スループットな蓄積と高速な検索はトレードオフの関係にあるため、その両立にはデータ構造に工夫が必要となる。

また、大量データを蓄積するために、汎用サーバを組み合わせて全体として大容量ストレージを形成する分散ストレージ技術の開発が、クラウドでのデータ蓄積に関連して盛んになっている。パケットデータの解析は、トラブル発生時には有用ではあるものの、適切な保存期間を前もって見積もることが難しく、スケールアウト性能に優れた分散ストレージであれば、小規模からスタートすることが可能となる。また、パケットデータには、時刻や送信元アドレスといったメタデータが付随している。パケットデータの解析にあたっては、メタデータによって対象となるパケット群を絞り込みながら解析を進めるが、メタデータによる検索機能を有したオブジェクトストレージであれば、パケットデータとメタデータを一元管理することで、データ管理コストを低減することが可能となる。

本論文では、パケットデータの高スループットな蓄積と、セッションの高速な検索を両立させる分散オブジェクトストレージ Nikaryo を提案する。パケットデータのような小さいサイズのデータを、古いデータの削除と検索を行いつつも高スループットに蓄積できる方式の提案を行うとともに、他の分散ストレージとの蓄積性能の比較、及び、典型的な検索パターンにおける検索性能の評価を行い、その評価結果を示す。

## 2. 課題

高速ネットワークを流れるパケットデータの蓄積・活用に向けた分散オブジェクトストレージを実現するにあたっては、以下の課題があげられる。

(A) 小中規模データの高スループット蓄積

一般にオブジェクトストレージは比較的大きなサイズ（数十 MB～）のデータをターゲットとしていることが多く、小中規模サイズ（～数 MB）のデータには焦点があたっていないことがある。しかし、パケットデータ蓄積・活用システムにおいては、蓄積と解析との両立のために、小中規模サイズのデータにおいてこそ、ネットワークスループットと同程度の高スループットで蓄積する性能が求められる。データサイズが大きすぎては解析時に使いづらく、小さすぎては蓄積

性能を出すことができない、というトレードオフから両者のバランスを最適化する必要がある。また、システム全体の総容量が同じであれば、オブジェクトのサイズが小さくなるほど、システム全体のオブジェクト数は多くなることを意味し、オブジェクト数が蓄積性能のボトルネックとならないような工夫が必要となる。

(B) データ削除の高速化

オブジェクトストレージではデータの蓄積に主眼が当てられ、データを削除することは重要視されていないことが多い。一方、パケットデータの蓄積においては、容量一杯まで蓄積した後は、古いデータを削除しつつ最新のデータを蓄積していくという運用が求められる。このため、データの削除に関しても蓄積と同等の性能が求められる。

(C) 蓄積性能を維持してのリアルタイム解析

パケットデータの解析においては、パケットの取りこぼしがないことが重要となる。そのため、常に蓄積性能を維持したまま、解析に必要なデータの検索、読み出しを高速に行う必要がある。

上記(A)～(C)をシステム全体として解決する分散オブジェクトストレージ Nikaryo を提案する。

## 3. 分散オブジェクトストレージ Nikaryo

ストレージサーバ間でオブジェクトを冗長化するにあたって、キャプチャマシンから直接複数のストレージサーバにオブジェクトを送信して冗長化すると、ネットワーク送信がボトルネックとなってしまう。また、パケットキャプチャシステムにおいては、オブジェクトの蓄積は定常的に高い負荷で行われることになるが、解析は ad hoc に行われる。そのため解析時のアクセスは負荷分散が可能であることが望ましい。さらに、小中規模サイズのオブジェクトを大量に蓄積するため、システム全体として非常に多数のオブジェクトを保持することとなり、オブジェクトの蓄積先を集中管理することは蓄積性能のボトルネックに繋がる。

Nikaryo は、ネットワークで接続された、HDD を搭載した複数のストレージサーバとプロキシモジュールから構成される分散オブジェクトストレージである。Nikaryo では、データ本体と各種メタデータをまとめたものをオブジェクトと呼び、オブジェクトは名前をベースにコンシステントハッシュを用いて各ストレージサーバに分散され、かつ、ストレージサーバ間で冗長化して保持される。各ストレージサーバは、オブジェクトの HDD への書き込みや、他のストレージサーバとの連携を制御する。また、ハッシュ空間において自分が分担する領域を他のストレージサーバと調整して決定する。Nikaryo へのアクセスはプロキシモジュールを経由して行う。プロキシモジュールは、オブジェクトがどのサーバで保持されているか（初めて書き込む際には、オブジェクトをどのサーバに送るべきか）というハッシュ空間の分担表を、前もってストレージサーバに問い合わせてキャッシュしておき、以後はアクセスに失敗した場合に、再度ストレージサーバに問い合わせて分担表の更新を行う。

また、検索のためのインデックス情報を保持するために、各オブジェクトにはバケツという上位概念を設ける。図 1 に概念図を示す。バケツとオブジェクトの関係は、ディレクトリとファイルに相当し、バケツの配下には複数のオブジェクトが存在する。そして、バケツには、配下にあるオブジェクト群のメタデータのうち、検索に用いるメタデータを集約し、それらをインデックス化して保持している。また、バケツもオブジェクトと同様に名前をベースにコンシステントハッシュを用いて各ストレージサーバに分散され、かつ、ストレージサーバ間で冗長化して保持される。

サーバ間で冗長化されたオブジェクト（これをレプリカと呼ぶ）は、それぞれ順位づけされた役割（Primary, Secondary, Tertiary, …）を保持している。図 2 に示すように、プロキシモジュールから書き込む際には、鎖状に Primary → Secondary → Tertiary … と順々にオブジェクトが転送され、終端に達した時点で、今度は逆順に応答が返ってくる。また、各サーバではオブジェクトの書き込み要求を受け取ると、次のサーバにオブジェクトを転送するとともに、自分が所属するバケツに対してメタデータを転送する。受け取ったバケツにおけるインデックス追加処理も、書き込み処理と同様に鎖状に順々に処理される。一方、読み出し及び検索は、負荷分散のため、冗長化しているどのオブジェクトからも可能とする。

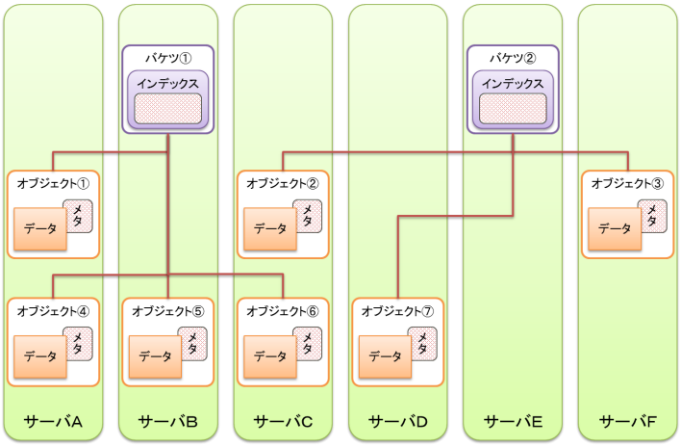



**図 1：バケツとオブジェクトの概念図**
**（レプリカ数＝1 の場合）**

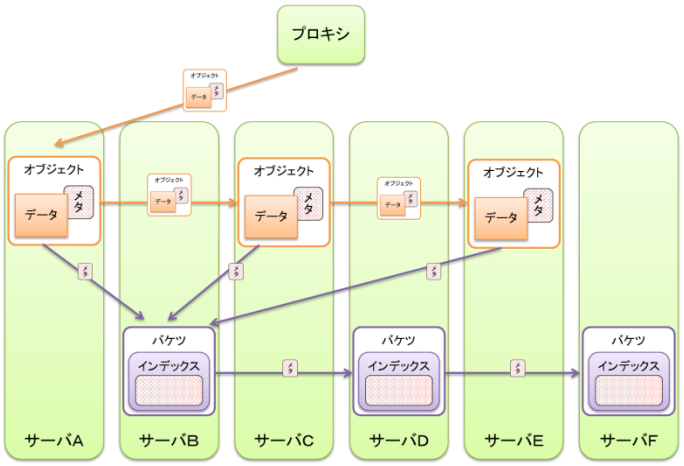



**図 2：書き込み及びインデックス作成の様子**
**（レプリカ数＝3 の場合）**

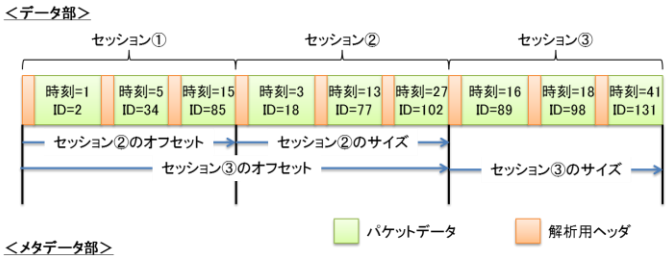
## 4. 高スループット蓄積と高速検索の実現方式

### 4.1. データ構造

Nikaryo が目指す分散オブジェクトストレージは、サイズの小さなパケットデータを高スループットで蓄積するとともに、想定される典型的な検索が高速で行える必要がある。小さなデータを高スループットで蓄積することのみが目的であれば、メモリ上にバッファリングを行い、シーケンシャルに HDD に書き込んでいくことを繰り返すことが考えられ、実際、そのような方式を用いているパケットキャプチャシステムもある。しかし、その方式では、解析する際に、時系列に蓄えられたパケットデータの中に読み出したいパケットが離散して存在することになってしまい、大量のランダム読み出しが発生し、高速な解析を実現できないばかりか、解析処理が蓄積性能に大きな影響を与える。

パケットデータ解析において典型的に用いられる検索のキーとして、時刻と送信元アドレスがある。このうち送信元アドレスによる検索への最適化として、Nikaryo では、全パケットを時系列順に書き込んでいくのではなく、送信元アドレスの適当な範囲（これを送信元グループと呼ぶ）でグルーピングして書き込む。これにより、送信元アドレスを指定した検索において検索範囲を限定することができ、高速検索を可能とする。また、送信元アドレスによるグルーピングに先だって、パケットのメタデータのうちの 5-tuple（送信先／元アドレス、送信先／元ポート番号、プロトコル番号）が一致するパケット群をセッションとして抽出する。そして抽出されたセッションに対しては、セッション単位でメタデータをまとめて保持する。これにより、パケット単位でメタデータを保持する場合と比較して、メタデータ容量の大幅に削減することができる。セッションのメタデータには 5-tuple の他に、セッションの開始／終了時刻やサイズ等が含まれる。また、セッションとして抽出されたパケット群に関しては、送信元アドレスによるグルーピングの際には、セッションの送信元アドレスを用いる。

パケットデータの蓄積・検索に向けたオブジェクト構造を図 3 に示す。各オブジェクトは同じ送信元グループに所属するパケット群から構成される。データ部は固定長で、パケットデータと解析用ヘッダから構成される。パケットデータは、セッション単位でまとめられ、さらにセッション内部では時系列順に並んでいる。セッション単位でまとめて蓄積することで、セッションベースの解析時に、蓄積したパケットデータ全体に渡って離散的に読み出すのではなく、特定箇所のみを読み出すこととなり、高速な解析処理を可能とする。また、パケットをバッファリングする領域のサイズは有限であるため、長時間続くセッションは、複数のオブジェクトにまたがって存在することになる。




| | 送信先 | 送信元 | プロトコル | 時刻 | サイズ | オフセット | パケットID |
|---|---|---|---|---|---|---|---|
| ① | 1.2.3.4:123 | 1.3.6.9:136 | TCP | 2～15 | 2800 | 0 | 2,34,85 |
| ② | 1.2.3.4:80 | 2.4.6.8.:80 | TCP | 3～27 | 50000 | 2800 | 18,77,102 |
| ③ | 2.4.6.8:100 | 1.3.6.9:138 | UDP | 16～41 | 4500 | 52800 | 89,98,131 |

**図 3：オブジェクトの例**

パケットデータの解析としては、特定のキーによる検索を何度か行って対象となるパケットデータを絞り込んだ後に、必要なパケットデータを読み出す、という使い方を想定する。その検索の際に HDD へのアクセスが発生してしまうと、パケットデータの蓄積性能に大きな影響を与えてしまう。Nikaryo では、メタデータをセッション単位に集約することで大幅にメタデータ容量を削減している。そこで、検索に用いるインデックスをメモリ上に配置することで、蓄積性能に影響を与えずに検索を実行することを可能とする。

各オブジェクトのメタデータは、バケツ単位で集約してインデックス化する。時刻と送信元アドレスを用いた検索への最適化として、Nikaryo では、オブジェクト内に含まれるセッションのうち最も早い開始時刻（これをオブジェクトの開始時刻と呼ぶ）と送信元グ

ループとから所属するバケツを決める。つまり時刻と送信元アドレスという 2 つの軸で分割したインデックスが作られる。

バケツ単位で検索処理及びインデックス作成を並列して行えるため、バケツ数を多くしておくことは検索性能を上げることにつながる。ただ一方で、あまり細かく分割すると、解析に必要な検索数が増えてしまうため､解析処理全体としては性能が下がってしまう。そのため、バケツ数を最適に設定する必要がある。

### 4.2. データアロケーション

HDD の蓄積性能が最も高いのは､データをシーケンシャルに書き込んでいった場合である。パケットデータの蓄積開始時には、新しいオブジェクトをシーケンシャル書き込んでいくことで最高の書き込みスループットを引き出せる。しかし、問題は HDD の容量一杯にデータを書き込んだ後、古いデータを削除しつつ新しいデータを書き込む場合である。古いデータの削除に伴う領域管理情報の更新で HDD にアクセスが発生してしまうと、データ書き込みのシーケンシャル性が阻害され、蓄積性能に影響が出てしまう。

そこで、Nikaryo では、HDD の領域管理のための情報をメモリ上にビットマップで保持する。オブジェクトのデータ部を固定長にしているのは、このビットマップ管理を簡素化するためである。また、メタデータ部に関しては、複数のオブジェクトのメタデータをまとめて、データ部と同じ固定長に近い大きさにしてから HDD 上に記録する。

パケットデータ蓄積においては、古いオブジェクトを削除しながら、新しいオブジェクトの蓄積を進めていく。オブジェクトを書き込む際の HDD 領域の割り当ては、空き領域の分布に関わらず、アドレスが増加する方向に割り当てていき、終端に達したら再度先頭から領域割り当てを行う。

### 4.3. 検索方式

先に述べたように、インデックスはオブジェクトの開始時刻と送信元グループとで分割して作成されているため、検索条件に時刻か送信元アドレスが含まれている場合、検索対象の絞り込みが可能である。検索時には、まず検索条件から検索対象となるバケツ群を抽出する。その後、抽出したバケツ群に対して並列に検索クエリを投げ、返ってきた結果をマージして、全体の検索結果を生成する。

## 5. 評価

提案する分散オブジェクトストレージ Nikaryo を用いたパケットデータの蓄積・活用性能の評価として、他の分散ストレージ（Apache HDFS[7], OpenStack Swift[8]）との蓄積性能の比較、及び、典型的な検索パターンにおける検索性能の評価を行った。

### 5.1. 実験環境

評価実験は、キャプチャマシン 1 台とストレージサーバ 30 台及び検索サーバ 1 台を 10Gbps のネットワークで接続した実験システムで行った。

キャプチャマシンとしては、表 1 に示すように、8 コア/16 スレッドの Intel Xeon E5-2670 を 2 ソケットと 256GB の主記憶を搭載したサーバを用い、ストレージサーバとしては、表 2 に示すように、4 コア/8 スレッドの Intel Xeon E3-1270 を 1 ソケットと 16GB の主記憶、及び 1TB の SATA HDD を 4 台搭載したサーバを用いた。また、検索サーバとしては、ストレージサーバと同じサーバを用いた。それらが Cisco Nexus 5548 を経由して､互いに 10Gbps のイーサネットで接続されている。また､各サーバでは OS として 64bit 版 CentOS 6.4 が動作している。

**表 1：キャプチャマシンの諸元**

| | |
|---|---|
| Model | Fujitsu PRIMERGY RX300 S7 |
| CPU | Intel Xeon E5-2670 × 2 |
| Memory | 256GB |

**表 2：ストレージサーバ／検索サーバの諸元**

| | |
|---|---|
| Model | Fujitsu PRIMERGY RX100 S6 |
| CPU | Intel Xeon E3-1270 |
| Memory | 16GB |
| HDD | Seagate Constellation2 SATA 1TB × 4 with RAID0 |
| RAID Card | LSI SAS1064e |

### 5.2. ストレージサーバの HDD 性能

各ストレージサーバでは、RAID カードを用いて、4 台の SATA HDD で RAID0 を構成している｡その RAID0 ボリュームに対して、I/O サイズを変えて、シーケンシャルリード／ライト、及び、ランダムリード／ライトの性能を測定した｡シーケンシャル性能は IOzone[7] を用いて測定し、ランダム性能は、非同期ダイレクト I/O を用いて HDD 全領域に対してランダムにアクセスを発行して測定した。図 4 に示すように、シーケンシャルライト性能は I/O サイズに関わらず 365MB/s 程度となる。仮に、分散ストレージを構成する全てのストレージサーバが、常時シーケンシャルに HDD に書き込むという理想的な状態で動いたとすると、レプリカ数が 1 であれば 4 台程度、レプリカ数が 3 であれば 10 台程度で、ネットワークの性能限界である 10Gbps に達する。ランダムライト性能は I/O サイズに応じて変

動する。I/O サイズが大きければ性能は高いが、解析時のオーバヘッドが大きくなる。以降の評価では、データサイズとして 4MB を用いる。

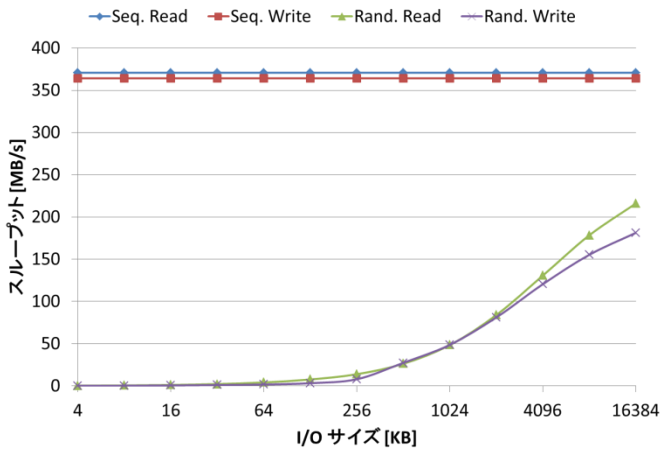



**図 4：ストレージサーバの HDD 性能**
**（HDD×4 with RAID0）**

### 5.3. 蓄積性能比較

パケットデータ蓄積において定常状態となる古いデータを削除しながら新しいデータを蓄積している状態の蓄積性能に関して、オープンソースの分散ストレージである Apache HDFS 及び OpenStack Swift との比較を行った。HDFS は CDH4.3.1 を、Swift はバージョン 1.7.4-2 を用いた。負荷マシンとしては前述のキャプチャマシンを用い、負荷マシン上で、Nikaryo 及び Swift の測定ではプロキシモジュールを、HDFS の測定では NameNode を動作させた。

Nikaryo では、検索範囲を限定するために送信元アドレスに応じて複数のバケツを作成する。それを模して、各ストレージとも、事前に 256 個のバケツを作成しておき、ランダムにバケツを選択して 4MB のオブジェクトを書き込むことを繰り返した。そして古いオブジェクトの削除を並行して行った際の蓄積性能を測定した。具体的には、①オブジェクトの書き込み開始、②1800 秒後に、作成後 60 秒以上経過しているオブジェクトの削除を開始、③削除を開始してから 180～240 秒後の蓄積性能の平均を算出、といった手順で行った。

図 5 にレプリカ数が 3 の場合の蓄積性能を示す。Nikaryo は理論性能に近い蓄積性能が出ており、ストレージサーバ台数が 16 台で性能限界の 10Gbps に達している。一方、ストレージサーバ 16 台における HDFS と Swift の蓄積性能は、HDFS が Nikaryo の 1/2 程度、Swift が 1/3 程度であった。

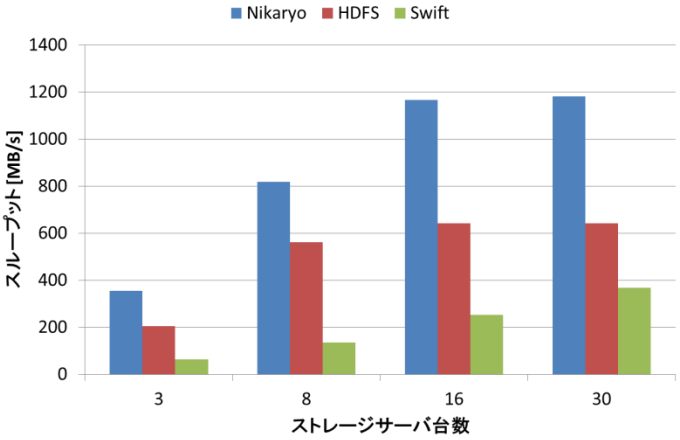



**図 5：蓄積性能の比較**
**（レプリカ数=3、オブジェクトサイズ=4MB）**

Swift はレプリカの作成方式が Nikaryo や HDFS と異なり、プロキシモジュールからレプリカを蓄積する複数のストレージサーバに対して直接書き込みを発行する。そのため、レプリカ数が 3 であれば、1 台の負荷マシンから可能な蓄積性能の限界値が Nikaryo や HDFS と比較して 1/3 となる。そこで、図 6 にレプリカ数を 1 とした場合の蓄積性能を示す。Nikaryo は理論性能に近い蓄積性能が出ており、ストレージサーバが 3 台の時点で性能限界の 10Gbps に達している。また HDFS もストレージサーバ 3 台の時点でレプリカ数が 3 の場合と同じ蓄積性能に達する。一方 Swift だが、レプリカ数を 1 とすることでレプリカ数が 3 の場合と比較して性能の上限値は向上するものの Nikaryo と比較すると 1/2 程度の蓄積性能となった。

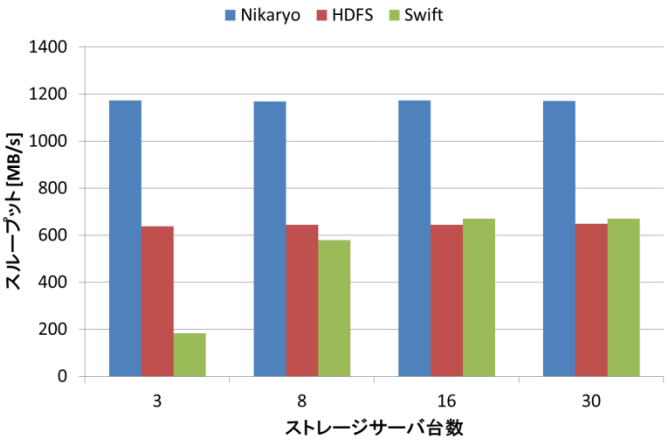



**図 6：蓄積性能の比較**
**（レプリカ数＝1、オブジェクトサイズ=4MB）**

HDFS は、Nikaryo と比較すると 1/2 程度の蓄積性能で限界に達しているが、これは NameNode の CPU ネックとなっているためである。HDFS は書き込む際に NameNode に問い合わせてから書き込み先を決定するため、オブジェクトサイズが小さいほど NameNode への問い合わせ回数が増える。そこで、オブジェクトサイズを 64MB として測定した結果を図 7 に示す。オブジェクトサイズが 4MB だったときの限界がなくなり、Nikaryo と同様に性能限界の 10Gbps に達している。

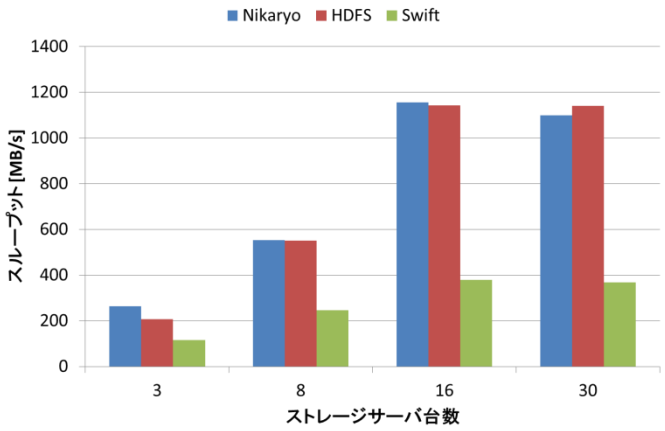



**図 7：蓄積性能の比較**
**（レプリカ数＝3、オブジェクトサイズ=64MB）**

### 5.4. 検索性能

実際にキャプチャ処理を行いつつ、パケットデータを蓄積しながら検索を行った場合の性能を測定した。キャプチャするトラフィックは、Agilent 社の N2X を用いて作成した。256 個の送信元グループに対して、それぞれ 1 本の TCP セッション（合計 256 セッション）が全体として 10Gbps で流れているトラフィックをキャプチャマシンにてキャプチャし、それを Nikaryo に蓄積していき、ある程度時間が経過してから検索処理を行い、そのレスポンス時間を測定した。具体的には、キャプチャを開始してから 600 秒間経過した後、その時刻から 600～599 秒前の 1 秒間に含まれるセッションの検索を、送信元アドレス（SA）を指定しないケースと指定したケースで 10 回ずつ行い、そのレスポンス時間の平均を求めた。

ストレージサーバ 16 台を用いて、レプリカ数を 3 にし、パケット長を 256～1280 Byte の間で変えて測定した結果を図 8 に示す。左軸が蓄積性能を、右軸が検索性能を表す。パケット長が小さいほど、セッション抽出等に必要な処理が増加する。そのため、パケット長が 256 Byte の場合には、キャプチャマシンの CPU がボトルネックとなり、キャプチャ処理でパケットの取りこぼしが発生してしまった。その結果、蓄積するパケットデータ量が低下した。パケット長が 1024 Byte では取りこぼしなくキャプチャできるようになり、10Gbps の蓄積負荷がかかるようになった。10Gbps の蓄積負荷がかかるパケット長が 1024 Byte 及び 1280 Byte においては、送信元アドレスを指定してセッションの検索を行うと、20～30 ミリ秒程度で検索結果を入手できた。また、送信元アドレスを指定せずにセッションの検索を行った場合には、800～900 ミリ秒程度で検索結果を入手できた。

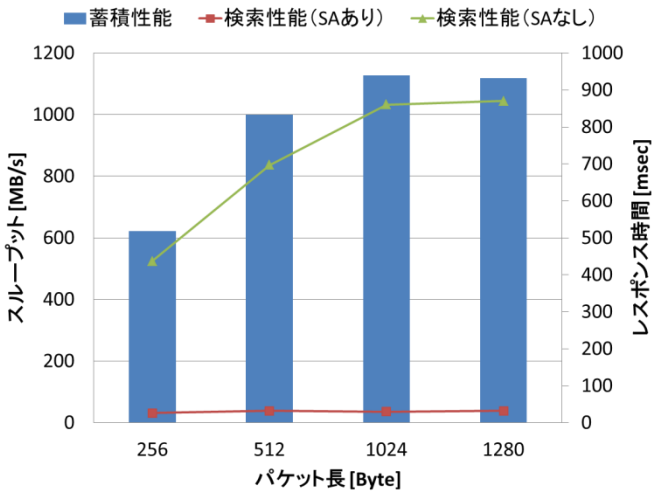



**図 8：パケットデータ蓄積時における検索性能**

## 6. 関連研究

Morariu ら[3]は、分散ハッシュテーブルを用いた P2P 型の分散ストレージを用いたパケットキャプチャシステムを提案した。パケットの送信元アドレス等を用いた蓄積先サーバの選定、サーバ内での検索に特化したデータ構造を適用することで、高速な検索が実現できることを示した。Lee ら[2]は、Hadoop を用いる方法を、Fusco ら[4]は、ビットマップ・インデックスを用いて、高速にパケット検索を可能とする方法を示したが、いずれも蓄積性能に焦点を当てたものではない。

また、キャプチャマシン内でパケットデータを蓄積する方法として、塚原ら[1]は、HDD を搭載した汎用サーバにおいて、システムパラメータをチューニングすることでパケット蓄積性能を向上させる方法を示したが、高速ネットワークを流れるパケットデータを蓄積するには十分な性能とは言い難いものであった。また、Deri ら[5]は、SSD を搭載した汎用サーバにおいて、バッファ管理を効率化することで 10Gbps 近い性能を出すことを示したが、SSD を用いているため、長期間にわたる蓄積には高コストなシステムとなる。

提案手法は、パケットデータの高スループット蓄積と高速検索とを実現する分散オブジェクトストレージ

を、HDD を搭載した汎用サーバを用いて実現しようという点で、新しい取り組みとなる。

## 7. まとめ

本論文では、蓄積性能と検索性能を両立したパケットキャプチャシステムの実現に向けて、パケットデータをセッション単位で集約するとともに送信元アドレスに応じた小中規模サイズの固定長にまとめ、内包するセッションのメタデータとともにオブジェクトを生成し、それらをコンシステントハッシュを用いて分散させつつ、鎖状に書き込みを行って冗長化を確保するとともに、メタデータに関してオブジェクトの開始時刻と送信元アドレスの 2 つの軸で分割したインデックス作成を行う分散オブジェクトストレージ Nikaryo を提案した。Apache HDFS 及び OpenStack Swift と比較を行い、4MB のオブジェクトサイズで 16 台のストレージサーバに書き込んだ場合に、HDFS の 2 倍程度、Swift の 3 倍程度の蓄積性能を示した。また、10Gbps のトラフィックをキャプチャして蓄積している状態で、送信元アドレスを指定した検索では 20〜30 ミリ秒程度で、送信元アドレスを指定しない検索では 800〜900 ミリ秒程度でセッション検索が可能であることを示した。

今後は、提案方式の評価を進めるとともに、さらに高速なネットワークを流れるパケットデータの蓄積・活用を目指す。また、オブジェクトに付与されたメタデータを利用してのストレージサーバ側での分散処理にも取り組む。

## 参 考 文 献


[1] 塚原康仁, 遠峰隆史, 杉浦一徳, “汎用機器を用いたネットワークトラフィック解析を実現するパケットキャプチャリングシステムの最適化手法”, 情報処理学会研究報告, コンピュータセキュリティ研究会, 2010-CSEC-48(7), pp.1-7, 2010

[2] Lee, Yeonhee, Wonchul Kang, and Youngseok Lee. "A hadoop-based packet trace processing tool." Traffic Monitoring and Analysis. Springer Berlin Heidelberg, 2011. 51-63.

[3] Morariu, Cristian, Thierry Kramis, and Burkhard Stiller. "DIPStorage: distributed storage of IP flow records." Local and Metropolitan Area Networks, 2008. LANMAN 2008. 16th IEEE Workshop on. IEEE, 2008.

[4] Fusco, Francesco, et al. "pcapIndex: an index for network packet traces with legacy compatibility." ACM SIGCOMM Computer Communication Review 42.1 (2012): 47-53.

[5] Deri, Luca, Alfredo Cardigliano, and Francesco Fusco. "10 Gbit line rate packet-to-disk using n2disk." Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on. IEEE, 2013.

[6] IOzone homepage: http://www.iozone.org/

[7] Apache Hadoop homepage: http://hadoop.apache.org/

[8] OpenStack homepage: http://www.openstack.org/