Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

KVS における動的性能伸張性の向上

堀内浩基[†1]　山口実靖[†1]

クラウドコンピューティングの普及により，大量の計算機資源を容易に入手できるようになった．これによりデータベースのスケーラビリティが重要視されるようになり，高いスケーラビリティを持ち動的に計算機台数を増減可能な KVS (Key Value Store)が注目を集めるようになった．本稿では，KVS の動的性能伸張性(ノード追加をして性能を向上させる処理に要する時間の長さ)について着目し，考察を行う．まず，動的性能伸張性の評価結果を示し，KVS に読込負荷がかかっている状況では性能伸張に多くの時間を要することを示す．そして，読み込み処理の性能に制限をかけ動的性能伸張性を向上させる手法を提案する．最後にその評価結果を示し，提案手法により，読み込み処理への大きな性能劣下なく，動的性能伸張性を大きく向上させることが可能であることを示す．

Improvement of Performance Elasticity in KVS

Kohki Horiuchi[†1]　Saneyasu Yamaguchi[†1]

With cloud computing systems, many computing resources are dynamically obtained with ease. In this environment, scalability of database management system can be important. For this purpose, KVS (Key Value Store), which is a simple database management system only with keys and values, is increasing its importance. Employing cloud computing and KVS is expected to enable very elastic database management system with which performance of database management system can be dynamically increased or decreased by adding or removing computing nodes. In this work, we have evaluated elasticity of VKS using Cassandra which is one of the most famous KVS. The results have shown that time to add a node into a working KVS system has significantly increased during processing get queries. For this issue, we have proposed a method for decreasing time to add a node by limiting performance of get queries of KVS. Our evaluation has demonstrated that the proposed method can significantly decrease time to add a nodes without large impact on performance on get query processing in KVS.

1. はじめに

クラウドコンピューティングの普及に伴いデータベース管理システムのスケーラビリティの確保が重要視され，この解決策として Key-Value Store (KVS) が注目されている．KVS は Key と Value のみで構成されるシンプルなデータ構造のデータベースであり，高いスケーラビリティがある．また，KVS は実行時にノード数の増減を行うことが可能であり，クラウド環境の様に使用計算資源量を容易に増減できるシステム上で KVS を動作させれば，性能を動的に伸縮可能であるデータベース管理システムを構築できると期待されている．

我々は，実行中の KVS システムにノードを追加して動的に性能を拡張させることに着目し，文献[1]にて動的性能伸張性の評価と動作の解析を行った．そして，読込負荷(get 要求処理)中にノード追加(join 処理)を行うと，ノード追加時間(join 時間)が大幅に増加してしまうことを示した．

本稿では，読込負荷の性能に制限をかけて，join 時間を短縮させる手法を提案し，提案手法適用時の join 時間と get 応答時間の評価を行う．

2. KVS

2.1 KVS の特徴

KVS とは，Key と Value のみで構成される単純なデータベース管理システムである．データベース内に Key と Value の組を保存することができ，Key を指定してそれに対応する Value を得ることができる．構造が単純であるため分散環境への適用が容易であり，高い規模拡張性が得られることが多い．逆に，多くの場合提供される機能が豊富ではなく，また保証される一貫性が高くないため，利用者がこれらを考慮した運用をすることが求められる．

代表的な KVS の実装の 1 つに Cassandra[2]がある．

2.2 Cassandra

Cassandra は Dynamo[3]の分散ハッシュテーブルと BigTable[4]のデータモデルを併せ持った Eventually Consistent な分散システム構造の KVS である．図 1 の様に，Cassandra を構成する各ノードはトークンと呼ばれるハッシュ値を持ち，リング状のハッシュ空間にトークンをもとに配置される．各ノードは，ハッシュ値が自身のトークン値以下でかつ直前ノードのトークン値より大きい範囲を担当する．データを保存または検索する際は，Key をハッシュ関数にかけそのハッシュ値から当該データの担当ノードを特定する．

稼働中の KVS システムに新規ノードを追加した場合，新規ノードの担当トークン範囲にあたるデータが，既存ノードより新規ノードに転送される．図 2 に動的ノード追加に伴う担当範囲とデータの移動を示す．同様に，稼働中の KVS システムからノードを削除した場合は，脱退ノードの担当トークン範囲のデータが新しく担当することになるノードに転送される．本稿では特に，新規ノード追加の際に

†1 工学院大学
Kogakuin University

Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

生じる「既存ノードから新規ノードへのデータ転送」に着目して考察を行う．

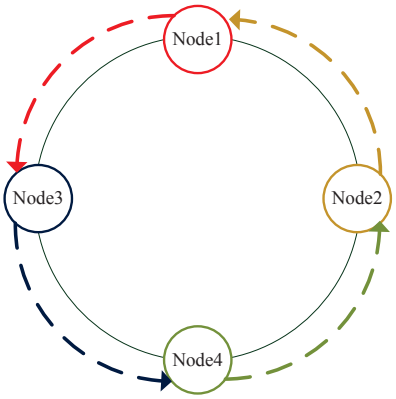

図 1　ノード配置

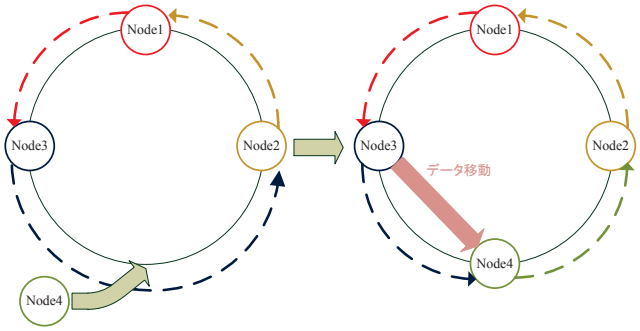

図 2　ノード追加に伴うトークン範囲・データ移動

3. 性能伸張性の評価

KVS では，実行時にノードを追加しシステム規模や性能を動的に拡張することができる．本章では，代表的な KVS 実装である Cassandra を用いて KVS の規模拡張(ノード追加)性能の評価を行う．

Cassandra では，ユーザがノード追加要求(join 要求)を発行すると，ノードはまず joining 状態でシステムに加わり，join 処理終了後にノードは normal 状態となる．本稿では，「join 要求を発行した時刻」から「(joining 状態を終えて)normal 状態になった時刻」までの時間を「join 時間」と定義する．

以下の節にて，各条件下における測定結果を示す．全ての実験において，使用した KVS は Cassandra 1.0.7，レプリカ数は 1，使用した OS は Linux 2.6.18.8，Value サイズは 16[KB]，実験環境は図 3 の通り，使用計算機の仕様は表 1 の通りである．また，次節以降で示される読込処理とは一様分布乱数で key を指定しその value を get 要求により取得する処理であり，書込処理とは一様分布乱数で key を指定しその value を update 要求により書き込む処理である．

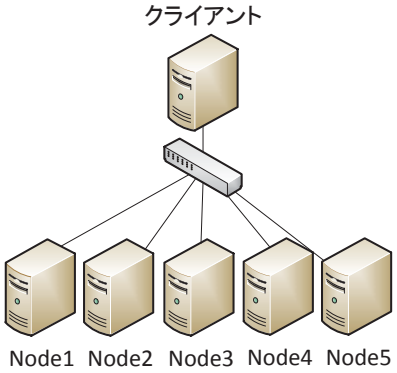

図 3　測定環境

表 1　使用計算機の仕様

クライアント,Node1,Node2	
CPU	Intel Core i3
Mem	2[GB]
HDD	150[GB]
NIC	Gigabit Ethernet

Node3,Node5	
CPU	Intel Celeron
Mem	2[GB]
HDD	150[GB]
NIC	Gigabit Ethernet

Node4	
CPU	AMD Athlon
Mem	3[GB]
HDD	150[GB]
NIC	Gigabit Ethernet

3.1 ノード台数と性能

KVS システムを構成するノード数と KVS システムの性能の関係の評価を行う．

まず，総データサイズ 10[GB]，データ均等配分，ノード数 1 から 5 における KVS システムの性能を図 4 に示す．性能は，16 スレッドで 1 秒間あたりに処理できる get 要求の数である．データは各ノードで均等に配分されており，ノード数 n における各ノードの保持データサイズは 10/n[GB]である．

次に，データサイズ 16[GB]，データ不均等配分，ノード数 1 から 5 における性能を図 5 に示す．ノード数 n 台における各ノードの保持データサイズは 2 台目から n 台目までの n-1 台に関しては全データの 20[%]であり，1 台目のノードは残りのデータをすべて保持する．例えば n=4 の場合，2 台目，3 台目，4 台目のノードがそれぞれ全体の 20[%]データを持ち，1 台目のノードが残りとなる全体の 40[%]のデータを持つ．この不均等配分の実験は，次節以降で述べるノード追加実験に対応している．

図 4，図 5 よりシステム性能はノード台数に対して単調に増加していることが分かり，ノードの追加により KVS システムの性能を増加させることが可能であることが確認できる．

Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

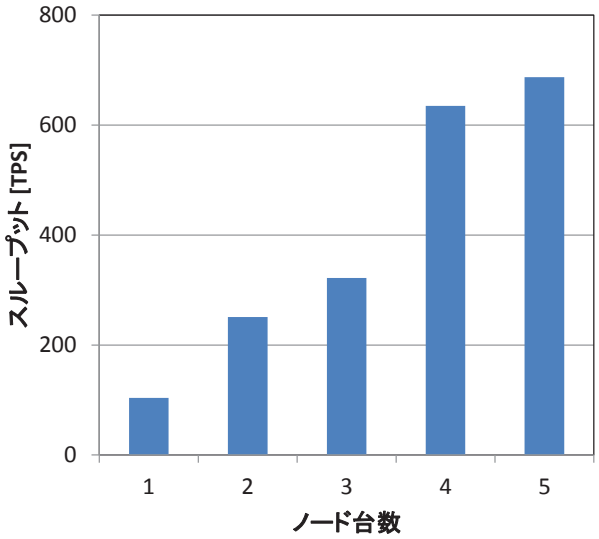

図 4　ノード数と性能の関係(均等配分)

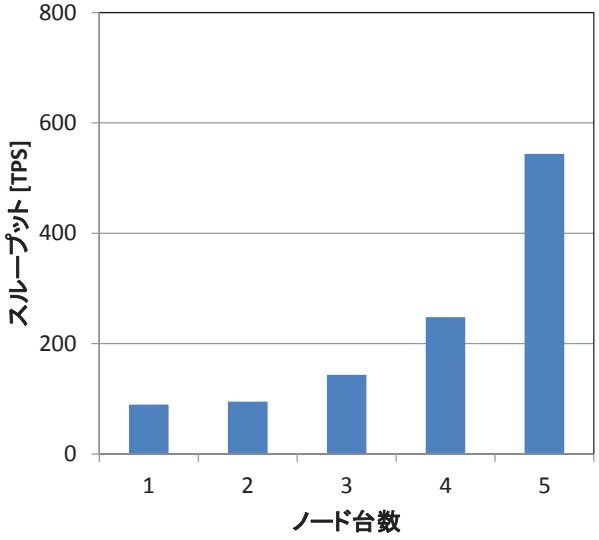

図 5　ノード数と性能の関係(不均等配分)

3.2　join 処理中の KVS システム性能

次に join 処理中の性能と join 中以外における性能の比較を行う．1 台のノードで構成される KVS システムに 1 台ずつ 4 台のノードを追加し，ノード台数を 5 台にまで増加させたときの join 処理中および join 処理中以外の性能を図 6 に示す．

図より，join 処理中の性能は join 処理中以外と比較して低く，性能向上のために稼働中の KVS システムに動的にノードを追加させると一時的に KVS システムの性能を低下させることが分かる．よって，join 時間の短縮や join 処理による KVS の処理性能の低下の軽減が重要であると考えられる．

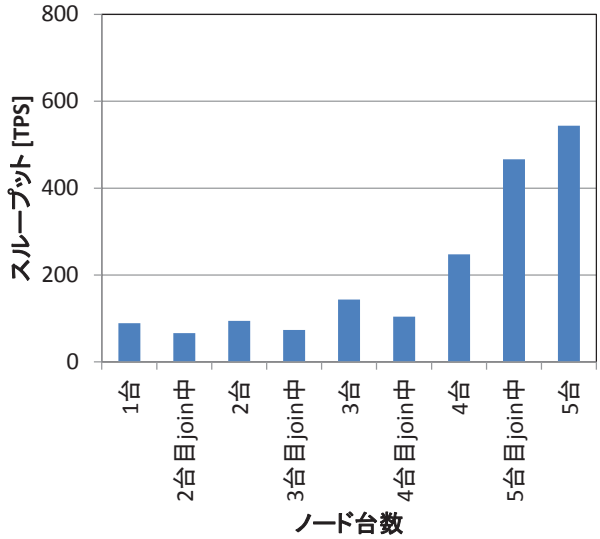

図 6　動的ノード追加におけるノード数と性能の関係

3.3　データサイズと join 時間

既存ノード数が 1，データベースの総サイズが 0[GB]から 16[GB]，join 処理中の KVS の負荷状態が負荷なし，読込負荷あり，書込負荷ありにおける 1 ノードずつ順に 4 ノード join させたときの各ノードの join 時間を図 7，図 8，図 9 に示す．ただし，初期状態では，既存の 1 ノードがトークン範囲 100%を担当しており，新規追加ノードはそれぞれ 20%を担当範囲として join を行う．

実験結果より，join 時間は 60 秒を下回らないこと，データサイズが増加するのに従い join 時間も増加する傾向があること，読込負荷(KVS get 処理)が存在すると join 時間が大幅に増加してしまうことがわかる．join 時間が 60 秒を下回らない理由は，Cassandra の join 実装内に 30 秒の sleep 処理が 2 個含まれているためである．

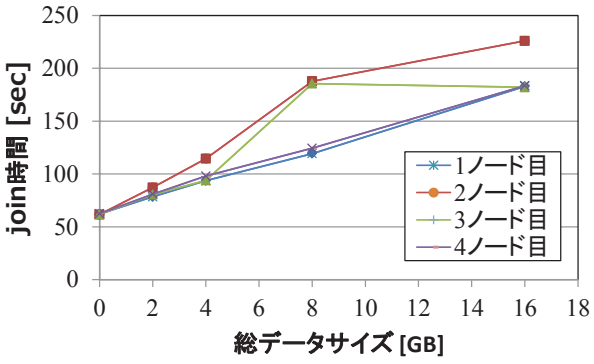

図 7　負荷なし時における総データサイズと各ノードの join 時間の関係

Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

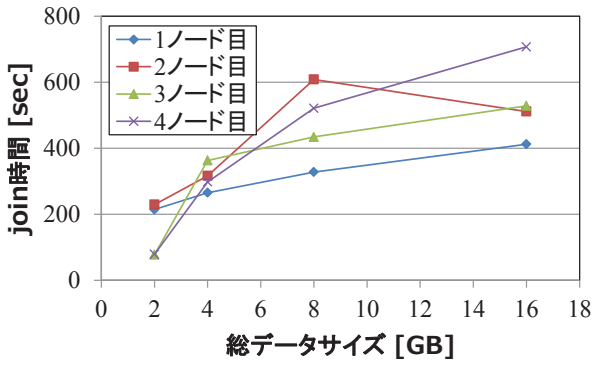

図 8　読込負荷時における総データサイズと各ノードの join 時間の関係

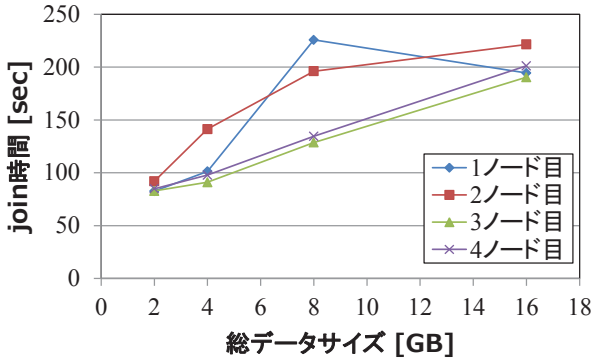

図 9　書込負荷時における総データサイズと各ノードの join 時間の関係

3.4　トークン範囲と join 時間

既存ノード数が 1，データベースの総サイズが 16[GB]，追加ノード数が 1 台，追加ノードのトークン範囲が 1[%]から 32[%]，join 処理中の KVS の負荷状態が負荷なし，読込負荷あり，書込負荷ありにおける join 時間を図 10 に示す．実験結果より，60 秒の sleep 時間を除くと join 時間はトークン範囲とほぼ比例することが分かる．また，前節同様に読込負荷中は，join 時間が非常に長くなることが分かる．

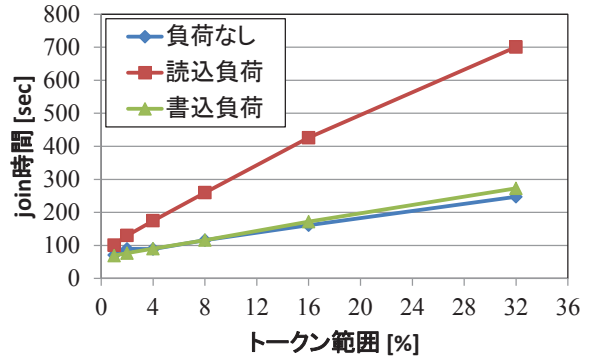

図 10　トークン範囲と join 時間の関係

3.5　並列 join ノード数と join 時間

既存ノード数が 1，データベースの総サイズが 16[GB]，並列 join ノード数が 1 から 4，join 処理中の KVS の負荷状態が負荷なし，読込負荷あり，書込負荷ありにおける join 時間を図 11 に示す．ただし，複数ノード並列 join の場合は，join 時間が最も長いノードの join 時間を「並列 join の join 時間」とした．

図より，並列 join ノード数を増加させることにより join 時間が比例以上の速度で増加し，複数台を並列して join させるより 1 台ずつ順に join を行った方が短い時間で join が完了できることが分かる．

これらの結果より，読込負荷中におけるノード追加には多くの時間を要することが分かった．

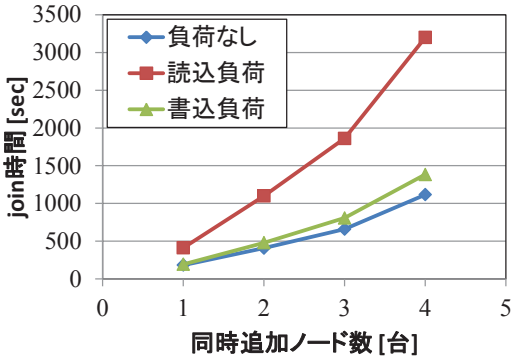

図 11　複数ノード並列追加と join 時間の関係

3.6　join 処理における発行 I/O

KVS の負荷状態が負荷なし，読込負荷あり，書込負荷ありにおける 1 台ノード join 時の disk I/O を図 12，図 13，図 14 に示す．

これらの図において，各プロットは HDD へのアクセス要求の発生を示しており，横軸はアクセス要求の発生時刻，縦軸はアクセス要求のアクセスアドレスである．

負荷がない状態の図 12 に着目すると，72[GB]付近から 74[GB]付近など，join 処理のためにデータベース内のデータを連続して読み込んでいることが分かり，これがシーケンシャルリードとして効率的に処理されていることが分かる．これに対して読込負荷がある図 13 の状況では，join 処理のためのデータベース内のデータの連続読込が get 要求の発生により細かく分断され，効率の悪いランダムアクセスとなっていることが分かる．書込負荷がある図 14 では，図 12 とほぼ同様に join 処理のためのデータ連続読み込みが効率のいいシーケンシャルリードとして処理されていることが分かる．

Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

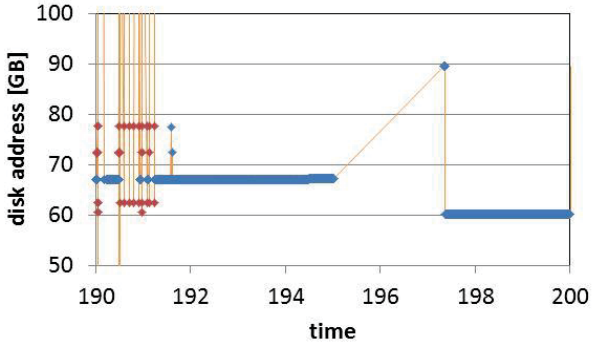

図 12　負荷なしの 1 台ノード join における disk I/O

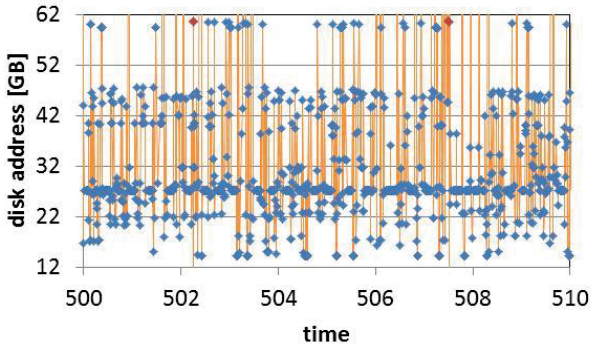

図 13　読込負荷の 1 台ノード join における disk I/O

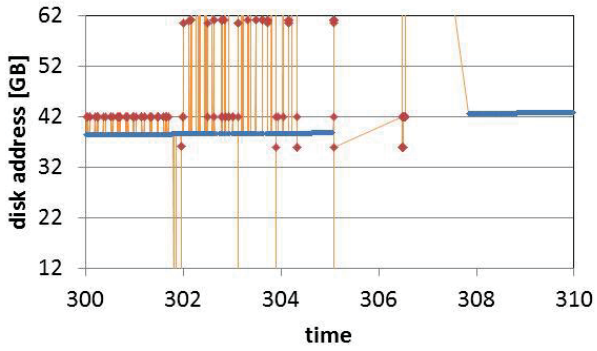

図 14　書込負荷の 1 台ノード join における disk I/O

4. 伸張性向上手法

読込負荷中におけるノード追加に要する時間を短縮させるために，get 要求の処理に遅延時間を加え，join 処理への影響を軽減させる手法について考察する．

4.1 システムの構成

KVS のクライアントからの get 要求の処理に遅延を発生させるために，図 15 のシステムを構築して性能を測定した．図中の Dummynet はネットワークエミュレータであり，本エミュレータにて ethernet パケットの転送に遅延を発生させた．

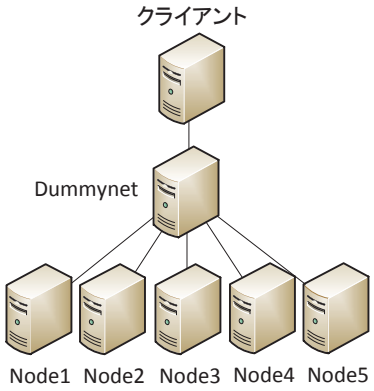

図 15　測定環境

4.2 性能評価

人工遅延時間を 0[ms]から 256[ms]まで変化させ，読み込み負荷中における 4 ノード並列 join の join 時間を評価した．測定結果を図 16 に示す．

図より遅延時間が増加すると join 時間が短縮していることが分かる．また，遅延時間 32[ms]以下では人工的な転送遅延時間の挿入による KVS システムの性能劣下(応答時間の増加)がみられず，システム性能の大きな劣下などの負の影響を発生させることなく join 時間の短縮を実現していることが分かる．人工遅延時間 64[ms]以上では，さらなる join 時間の短縮が実現されているが，get 要求の応答時間の増加も見られ，join 時間とシステム性能はトレードオフの関係となっていることが分かる．

次に，人工遅延時間 0[ms]，32[ms]，64[ms]において発行されたディスク I/0 要求(scsi コマンド)を図 17，図 18，図 19 に示す．

図より，join 処理のためのシーケンシャルリードの分断が，人工遅延時間 64[ms]においては相対的に小さく，人工遅延時間 0[ms]において相対的に大きいことが分かる．これにより join 時間の短縮が実現されていることが分かる．

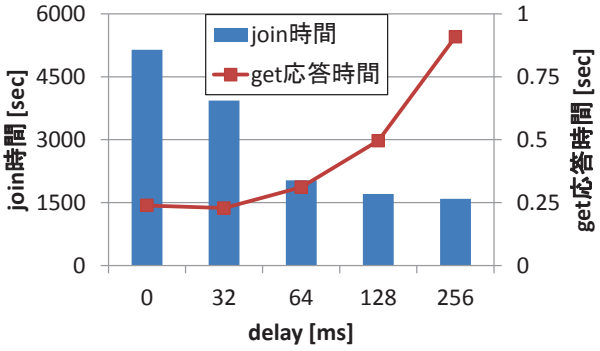

図 16　get 遅延と join 時間，get 応答時間の関係

Vol.2013-DPS-154 No.10
Vol.2013-CSEC-60 No.10
2013/3/14

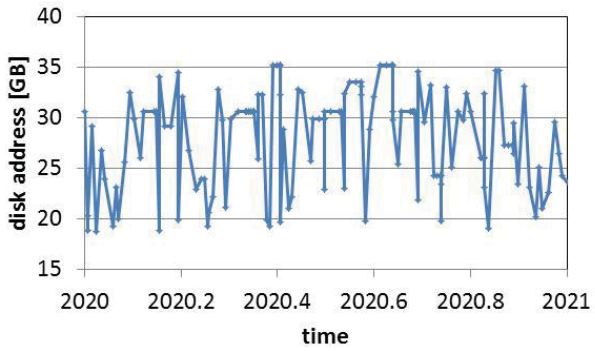

図 17　遅延時間 0[ms]における disk I/O

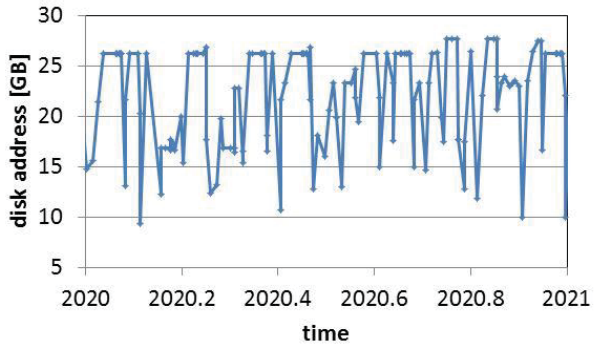

図 18　遅延時間 32[ms]における disk I/O

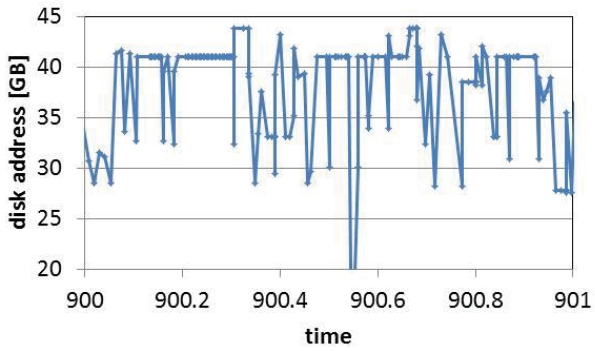

図 19　遅延時間 64[ms]における disk I/O

5. 関連研究

文献[5]においてKVSのdecommission処理(ノード削除処理)における，decommission時間と，decommission中のget処理性能の関係の調査および decommission 処理の性能劣下の提言手法の提案が行われている．また，性能評価により，提案手法により decommission 中の性能劣化の低減が可能であることが確認されている．

6. おわりに

本稿では，KVS の性能伸張性に着目し，代表的な KVS 実装 Cassandra の性能伸張性評価結果の紹介，伸張性向上手法の提案，提案手法の評価を行った．評価の結果，提案手法により，大きな KVS 要求性能の劣下なくシステム伸張性の向上が実現されていることが分かり，提案手法の有効性が確認された．

今後は，遅延処理の Cassandra システム内への実装を行っていく予定である．

謝辞　本研究は JSPS 科研費 22700039, 24300034 の助成を受けたものである．

参考文献

1)　堀内浩基, 山口実靖, ” KVS の動的性能伸縮に関する一考察”, 情報処理学会研究報告. EMB, 組込みシステム，2012 年 11 月
2)　Avinash Lakshman and Prashant Malik, “Cassandra- A Decentralized Structured Storage System”, LADIS 09, 2009
3)　Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,　Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels, “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP ’07, 2007
4)　Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes and Robert E. Gruber, “Bigtable: A Distributed Storage System for Structured Data”, IOSDI ’06 pages 205--218, 2006
5)　小林宙記，山口実靖，堀内浩基，"KVS におけるノード削除性能評価"，情報処理学会第 74 回全国大会，2012 年 3 月