従来のシステムと比べて柔軟性の高い key-value store の開発
—分散 KVS：Kyeeva—

1. 背景

近年クラウドコンピューティングが大きな盛り上がりを見せている. ここで, サービス提供者には「大量のコンピュータを用いた大量のデータの処理」が求められており, 当然の事ながら「大量のデータを蓄積し探索することができる」というシステムの需要が生まれている.

クラウドコンピューティングにおける基盤システムの一つであるキーバリュー型データストア（Key-Value Store, 以降 KVS)はその要望を満たすシステムで, リレーショナルデータベース（Relational Database, 以降 RDB）では不可能な, 大規模分散環境下でのスケールアウトを実現したデータストレージシステムである. KVS は RDB と異なり Get, Put, Remove などの比較的簡易なインタフェースを前提としていることで, 大規模分散環境であっても高い性能, スケーラビリティ, 可用性を備えている. 得に分散環境上で運用される KVS を分散 KVS と呼ぶ.

しかし KVS はそのシンプルなインタフェースの制約等から,「効率の良い範囲検索が難しい」「問い合わせ単位が一つのキーと値のペアに制限される」などの問題がある.

2. 目的

本プロジェクトでは, 複数の属性に対しての範囲検索を可能とする分散 KVS の開発を行うことを目的とした.

3. 開発の内容

3. 1 従来システムの発展と高速化

クリエータは本プロジェクト期間前から範囲検索可能な KVS を開発してきた。従来システムは並列指向関数型プログラミング言語である Erlang のみを用いて実装していた. 本プロジェクトでは, 各ノード間の通信処理やオーバーレイネットワーク構築等の処理等, Erlang によって簡潔かつ論理的に記述可能な部分を引き続いて Erlang で実装し, 速度が要求されるローカルノード内でのデータ検索処理等には C で実装を行うことで, システム全体の保守性とパフォーマンスを高度に両立させた.

3. 2 範囲検索

範囲検索とは「1000 円から 2000 円の間で購入できる商品の一覧が欲しい」のように, 範囲を指定するとその範囲内に含まれる全ての値を取得することを指す. 分散 KVS の中での非集中型の分散 KVS では, この範囲検索機能がほとんど実現されていない. 範囲検索を実現するにはデータの範囲等を集中管理

する管理ノードを用意するのが容易である. しかし, そうすると頻繁に管理ノードにアクセスが発生する上に, そのノードが単一障害点やボトルネックに成り得る危険性がある. また, 大規模な分散環境での動作も保証しなければならないために, 単純な実装では効率の良い動作を行うことができないというのも一つの原因となっている.

本プロジェクトではこの問題の解決手段として, P2P 構造化オーバーレイネットワークの一つである Skip Graph を実装する方法をとった. Skip Graph とはスキップリストと呼ばれるアルゴリズム(図 1)を参考にして生み出されたオーバーレイネットワークであり, 分散環境において効率的に指定された範囲を探索可能であるという特徴を備えている. したがってこれを用いることで, 非集中型でスケーラブルな分散 KVS の上で範囲検索を実現することが可能となった.

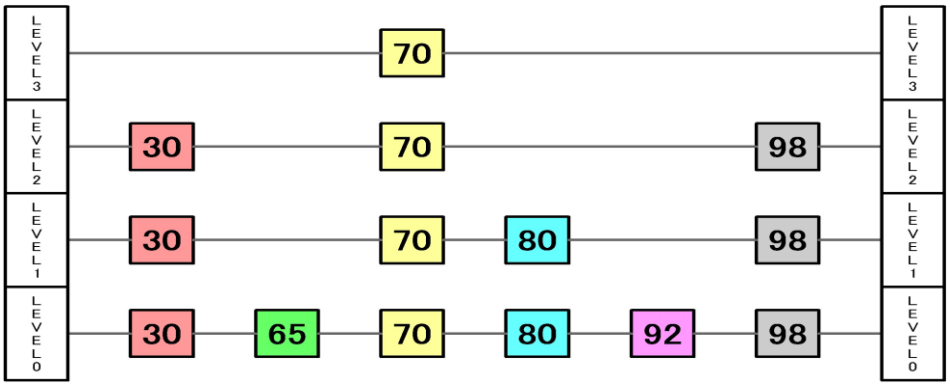

図 1. スキップリストアルゴリズム概念図

スキップリストは連結リストの階層になっており, 最下層は通常のソートされた連結リストである. また, 上層のリストに存在することのできる要素は乱数によって決定されるため, ちょうど急行列車のような働きを行う. 図 1 では N 層目のリストを Level N で表わしており, N が大きくなるほどリスト上の要素数は少なくなる. スキップリストにネットワーク通信を付け加えた物が Skip Graph と言えるため, このオーバーレイにおける範囲検索の実現方法はスキップリストの場合と本質的に同じである. したがってこれを用いることで, 非集中型でスケーラブルな分散 KVS の上で範囲検索を実現することが可能となる.

3. 3 多次元構造

単一アクセスを前提とした多次元構造を分散 KVS 上で実現させることに比べて, 範囲検索を前提とした多次元構造の実現は非常に困難なものとなっている. Skip Graph では一次元の範囲検索しか対応していない. そこで本プロジェクトでは, 複数の独立した Skip Graph を連携して使用することで, 多次元構造での範囲検索を可能にした. 本手法では図 2 に示すように, それぞれの属性と一

つの Skip Graph との対応づけを行う.

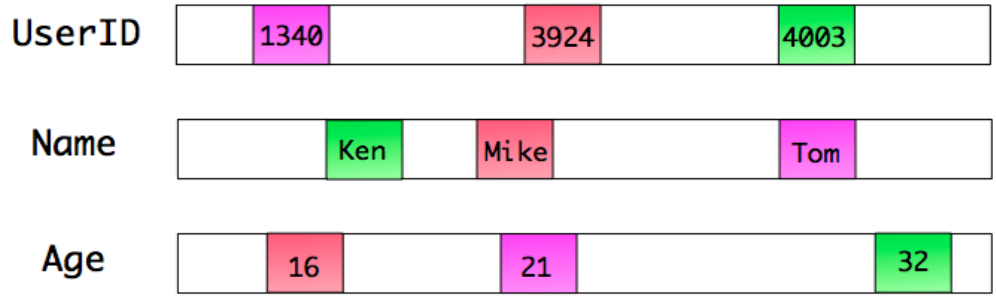

図 2. 複数の Skip Graph の組合せ

RDB におけるレコードの概念で考えると，後で処理するためには同じレコードに属するデータはそれぞれ関連付けされている必要がある．本システムでは，そのようなデータが同じ ID（Global Unique ID と呼ぶ，以降 GUID）を暗黙に持つように実装することで関連付けることとした．図 2 では同じ GUID を持つデータを同色で示している．

多次元構造の具体的な使用方法としては，「25 歳未満の人の名前を調べる」などが挙げられる．ここでは，年齢（Age）が「25 歳未満」という条件を満たすデータは 16，21 の二つが存在し，「25 歳未満の人の名前を調べる」という要求に沿った処理を行うには，二つのデータが属しているレコードの名前（Name）属性のデータをそれぞれ取得すれば良い．

実際には，属性の数だけ複数の Skip Graph を用意するのは実行効率を考えると好ましくないため，内部的には複数の Skip Graph を一つの Skip Graph に収める最適化を加えている．

3．4 負荷分散

Skip Graph はあくまで探索処理のためのオーバーレイネットワークであり，物理ノード間の関係には全く関与しない設計になっている．したがって，負荷分散等の物理ノードに深く関わる処理については別の仕組みが必要となる．本プロジェクトではこの解決方法として，物理ノードの管理用に新たに分散ハッシュテーブル（Distributed Hashing Table，以降 DHT）の層を追加した（図 3）．

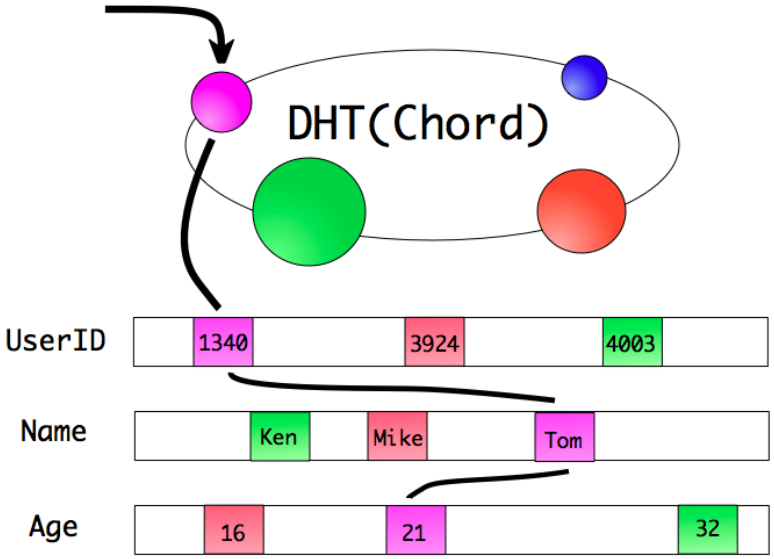

図 3. Skip Graph の上層に DHT 層を追加

この図に示すように，メッセージは DHT 層を通過した後に Skip Graph 層に到達するようになっている．ただしこの動作はデータの Put 時のみであり，通常の Get 処理などは物理ノード情報が不要なため，DHT を通過することなく Skip Graph 層からデータの取得を行うことができる．これにより DHT 層の追加によるオーバーヘッドは最小となっている．

本システムは StarBED 上の 40 台のノードを用いることで，十分実用的に動作することを検証できた．

4. 従来の技術（または機能）との相違

類似システムとして Google 社の BigTable，Facebook 社の Cassandra 等の分散 KVS が挙げられる．BigTable は範囲検索が可能であるが，管理ノードが必要という点で本システムと異なる．Cassandra は管理ノードが不要だが，範囲検索に向いていないという点で本システムとは異なるシステムとなっている．

5. 期待される効果

今後，高速化や各言語用ライブラリの作成を行うことで，広く利用可能なユーザが増えると思われる．特に Web サービスを開発している企業との連携が期待される．

6. 普及（または活用）の見通し

開発成果は既に GitHub にて公開中なので，誰でも自由にダウンロードすることが可能である．また，ドキュメントを整備し，一般のユーザが容易に使用できるようにする予定である．

7．クリエータ名（所属）

千々和 大輝(自由ヶ丘高等学校)

（参考）関連URL

http://github.com/daiki41ti/kyeeva