GridDB と InfluxDB を使用した

時系列データベースのパフォーマンス比較

April 4, 2018
Revision 1.9ja

目次

概要

Internet of Things（IoT）産業では 500 億以上のデバイスが運用され、その市場規模は 1 兆ドル以上になると予測されています。生成されるデータの多くは時系列（Time-Series）データであるため、このデータを活用するアプリケーションを開発する企業にとって、Time-Series データベース（TSDB）のパフォーマンスは重要な指標となります。フィックスターズは、市場でのマインドシェアを獲得しつつある 2 つの革新的なオープンソース TSDB -- GridDB（東芝）と InfluxDB（InfluxData 社）-- を比較しました。どちらも IoT に焦点を絞ったアーキテクチャをうたった製品です。

データベースの性能比較には、Amazon Web Services（AWS）上で動作する Yahoo Cloud Servicing Benchmark-Time Series（YCSB-TS）を用いました。時系列データベースの一般的な操作をすべてカバーするため、書き込み（Insert）、読み出し（read）、スキャン（scan）のパフォーマンスを比較しています。さらに、IoT ユースケースを想定した場合の重要なパラメータである、レイテンシやクエリなどの性能にも注目しました。パフォーマンス測定時の負荷としては 2 種類あり、一つ目のワークロードは読み取りクエリを発行してタイムスタンプを検索するもので、主に読み取りパフォーマンスのテストになります。二つ目のワークロードでは、時系列データに対する、スキャン検索、カウント操作、平均操作、合計操作、をすべて実行します。 これら 2 種類のワークロードが、1 億レコードと 4 億のレコードの 2 つの異なるデータセットで実施されています。

InfluxDB は数年の歳月を経て、IoT 市場において独自の地位を築きつつありますが、テストの結果では GridDB が、レイテンシとスループットの両面において InfluxDB を大きく上回っていることがわかりました。 GridDB のコンテナ（コンテナとはリレーショナルデータベースにおけるテーブルのようなもの）を巧みに使った革新的なアーキテクチャが、この結果をもたらしたと言えるでしょう。さらに、測定結果からは、GridDB の優れたスケーラビリティと一貫性も確認できました。こちらは、GridDB のインメモリ・アーキテクチャと、コンテナ内の ACID（Atomicity、Consistency、Isolation＆Durability）コンプライアンスによるものと言えます。

はじめに

データ管理では、時系列データは時間間隔で収集された一連の値として定義できます。時系列データの例としては、IoT センサデータ、ヘルスモニタ情報、太陽フレア追跡、イベント監視などがあります。従来の NoSQL データベースの多くは、大量の時系列データを処理するために十分なパフォーマンスとスケーラビリティがありません。その問題を解決するため、時系列データの処理に特化したデータベースが開発されまし

た。 GridDB や InfluxDB などの TSDB は、タイムスタンプ付きデータの格納、収集、取得、および処理に特化したデータベースです。 TSDB は、効果的なデータ圧縮、高い書き込みパフォーマンス、および高速範囲のクエリを提供するように最適化されています。これにより時系列データの処理を、従来のデータベースよりもスケーラブルで、信頼性高く、安価に提供することができるようになりました。

GridDB は、いわゆる NoSQL データベースです。信頼性向上のためハイブリッドクラスタアーキテクチャを採用し、インメモリ指向の分散型データベースとなっています。また、完全に ACID に準拠する数少ない NoSQL データベースの 1 つです。 GridDB では、「時系列コンテナ」にデータを格納することで、TSDB として使用できます。これらの時系列コンテナは、時間加重平均化および補間のような時間型インデックス付けおよびデータ関数を提供します。また、時系列コンテナには、期限切れのデータを効果的に解放する独自の圧縮ユーティリティが用意されています。 GridDB には、オープンソースライセンスと商用ライセンスとがあり、今回のベンチマークではオープンソースの GridDB “Community Edition”を使用しています。

一方、InfluxDB は TSDB であり、高い書き込み負荷とクエリ負荷を処理するために作られました。データを照会するための HTTP Representational State Transfer（REST）API、および InfluxQL と呼ばれる SQL のようなクエリ言語を使用できます。 InfluxDB はまた、複数のノードがストレージを処理すると同時にクエリを実行できる分散アーキテクチャを備えています。 さらに、時間構造マージ（TSM）ツリー・ストレージ・エンジンを使用して、高い書き込み速度を実現し、効果的なデータ圧縮を実行します。 InfluxDB には、オープンソース版と商用ライセンス版エンタープライズ版が用意されています。今回のベンチマークテストでは、InfluxDB “Open Source Edition”を使用しました。

YCSB-TS は、TSDB のテストに最適化されたベンチマークフレームワークで、一般的な NoSQL データベース向けのベンチマークフレームとして有名な YCSB （Yahoo! Cloud Service Benchmark）のソフトウェアフォークです。オリジナルの YCSB と同様、Java で書かれています。 YCSB-TS は、時系列範囲選択をサポートし、time-domain 関数を使用し、時系列データベースに固有の作業負荷オプションが追加されています。

テスト環境

Amazon AWS の設定

YCSB-TS ベンチマークテストは、CentOS 6.9 イメージをベースにした Amazon AWS EC2 インスタンスで実行されました。インスタンスタイプとしては、計算集約型の作業負荷に最適化されている C4.2xlarge モデルを使用しました。

AWS インスタンスには両方のデータベースをインストールしていますが、当然ながらテストは一度に 1 データベースずつ行っています。データベースサーバでは YCSB-TS クライアントも実行されています。テストの手順としては、まずデータベースサーバーを起動・接続し、YCSB クライアントを実行し、パフォーマンスデータとリソース使用量を収集・集計しました。

保存先に汎用（General Purpose）ブロックストレージタイプのデバイスを使ってしまうと、バースト機能の影響で測定結果が変動してしまうため、データはすべて、1000 IOPS, 50 GB の io1 Elastic Block Storage に保存されています。

AWS Instance Type	**C4.2xlarge**
Operating System	**CentOS 6.9**
CPU Processor	**Intel Xeon CPU E5-2666 v3**
vCPU Cores	**8**
Clock Speed	**2.9GHz**
Main Memory Size	**15GB**
Data Storage	**100GB io1 EBS with 1000 IOPS provisioned**

Table 1: AWS Instance Specifications

ソフトウェアのインストール

GridDB Community Edition バージョン 3.0.1 に関しては、公式の GridDB ウェブサイト (www.griddb.net) からダウンロードした RPM パッケージを、AWS インスタンスにインストールしました。 InfluxDB のバージョン 1.3.6 に関しては、InfluxData の Web サイト (www.influxdata.com) からダウンロードした公式 RPM パッケージをインストールしました。

YCSB-TS は、2017 年 10 月に公式の TSDBBench GitHub リポジトリからクローンしたものを使っています。YCSB-TS ディストリビューションはすべて Maven を使用してビルドされていました。 YCSB-TS の InfluxDB ドライバは、複数の Mesurements にデータセットを分散して配置できるように一部変更しました。また、オリジナルの YCSB フレームワーク向け GridDB ドライバを流用して、GridDB 用のカスタムデータベースコネクタを開発しました。

データベースの設定

GridDB

構成ファイルでは、gs_node.json の「storeMemoryLimit」を 6192MB（6GB）に増やし、小さなデータ・セットが storeMemory に収まるようにしました。 vCPU の数に合わせて並行性（Concurrency）を 8 に増やしました。

storeCompressionMode のフィールドを追加し、BLOCK_COMPRESSION に設定しました。GridDB の BLOCK_COMPRESSION では、メモリ内のデータをエクスポートして圧縮し、メモリの空き領域を解放します。これにより、大規模なデータセットを格納する際のストレージの使用量を最小限に抑えることができます。

InfluxDB

InfluxDB を使うにあたって懸念される問題として、カーディナリティの高いデータセットを使用する場合にメモリ使用量が高くパフォーマンスが低くなる点が挙げられます。 これを防ぐため、InfluxDB の設定ファイル、influxdb.conf を調整しました。 例えば http ロギングを無効にすることで、すべての POST およびインサート操作のログを保存しないようにし、ディスク使用量を減らしています。

また、Max-select-point と max-select-series を influxdb.conf で無効にし、何百万ものデータポイントを持つ時系列データに対する読み取りとスキャンのクエリが失敗しないように工夫しました。 他にも、大きなデータセットでパフォーマンスを向上させるため、cache-snapshot-write-cold-duration を 10 秒に設定し、インデックス・バージョンを tsi-1 に設定することも行いました。

InfluxDB の TSM ストレージエンジンは、デフォルトで圧縮が有効になっています。

テスト手法

設計

データロード時およびベンチマーク実行時のスレッド数は、128 スレッドとしました。 128 スレッドは通常、最も一貫性のあるパフォーマンスをもたらします。高カーディナリティコンテナ上で操作する場合、各データベースのスケーラビリティとパフォーマンスをテストするために、サイズの異なる 2 つのデータセットを使用することにしました。大規模なデータセットは、各データベースがデータをどのように圧縮するかを分析するのに効果的です。1 つ目のテストでは 1 億件のデータセット、2 つ目のテストでは 4 億件のデー

タセットを使用しました。レコードは、インスタンス内の vCPU の数と一致するように 8 つのコンテナまたはメトリックに分散されます。各コンテナまたはメトリックには、それぞれ 1,250 万または 5,000 万のレコードが含まれます。

1 億レコードのデータセットは合計約 5GB でメモリに残ります。 4 億レコードのデータセットは合計約 20GB となり、GridDB を使った場合、約 70%がディスク上に配置されます。 InfluxDB では、データをディスクへフラッシュするか、メモリに保持するか、という判断は、カーネルのディスクキャッシュ管理システムが行います。

YCSB-TS によって挿入されるすべてのレコードは、行キーとしてのタイムスタンプ、3 つの文字列タグ、およびメトリックの読み値である倍精度小数点値で構成されます。各文字列タグの長さは 10 文字で、合計 10 バイトです。これにより、各レコードは約 50 バイトになります。

Record Calculation:

$$(12\ byte\ timestamp) + (3 * (10\ byte\ string\ tag)) + (8\ byte\ double\ value) = 50\ bytes$$

すべてのデータフィールドは YCSB-TS によってランダムに生成されます。大規模データセットの場合、タイムスタンプの範囲はおおよそ 1～4 日に渡り、データの時間間隔は最低でもミリ秒はあるものとします。

Column Name	Column Type	Data Size	Example
Time	Timestamp (Row-key)	12 bytes	1439241005000
TAG0	String	10 bytes	“Wh64HSAlAU”
TAG1	String	10 bytes	“dXannLxHhW”
TAG2	String	10 bytes	“wTRxj0tNW9”
value	Double	8 bytes	5423.22

Table 2: YCSB-TS Database Schema

測定

一貫性を確保するため、測定は AWS インスタンス上で 3 回行い、各データセットのスループットとレイテンシを測定しました。スループットおよびレイテンシの値は、3 回の測定の中央値を採用し、数値の差が InfluxDB と GridDB の間のパフォーマンスの差異によるものであり、AWS インスタンスの性能に起因しないようにしました。

ヘッドノードは、「新規の、割り当てが解除された」状態から測定開始しています。 AWS インスタンスが起動すると、まずローカルの SSD およびデータベースディレクトリがマウントされます。 次に、AWS インスタンスは、各データベースからすべてのコンテナ、データ、ログを消去し、設定ファイルを読み込みます。そこから、InfluxDB または GridDB のいずれかを init スクリプトまたはデータベースコマンドによって起動します。

GridDB の場合、設定情報と統計情報は gs_stat コマンドで記録されます。 InfluxDB では Influx シェルを使用しました。

YCSB-TS ロード操作は、適切な insertstart、insertend、および recordcount パラメータ値を使用してクライアント・ノード上で実行されます。これらの値は、ワークロードの構成とデータセットのサイズに応じて調整されます。ロードフェーズが完了すると、下記の 2 つのワークロードのうちの 1 つが実行されます。

- ✓ ワークロード A: 読み取り専用
- ✓ ワークロード B: スキャン、カウント、平均、および合計操作

YCSB-TS のアーキテクチャと使用方法の詳細については、YCSB-TS GitHub のページを参照してください。
https://github.com/TSDBBench/YCSB-TS

集計

データベースのサイズ、CPU 使用量、メモリ使用量に関する統計は、ワークロードの終了後に Bash スクリプトを使用して取得されます。

YCSB-TS からのすべての出力は、bash スクリプトで処理され、ログファイルとして出力されます。測定後、 これらのログファイルのデータは、スプレッドシートプログラムで処理されます。YCSB-TS の実行中および実行後の両方で監視すべき重要な出力データは、データベースのディスク使用量と、そのデータが *data, log, wal* の各ディレクトリにどのように分散されるかです。 このデータを監視することで、データベースのサイズと圧縮手法を調べることができます。

CPU 使用率、メモリ使用量、ディスク使用量などの指標は、”top” やその他のコマンドの出力をログファイルにリダイレクトする bash スクリプトを使用して記録しました。

測定結果

Insert オペレーション

ベンチマークのタイムスタンプ範囲は 1～4 日です。各レコードの時間間隔は最低でも 1 ミリ秒あるので、データセットのサイズは、1 億～4 億レコードの範囲になります。 まず 1 億(100M) レコードの小さなデータセットを使用して、各データベースがメモリ内のデータセットでどのように動作するかをテストしました。 GridDB では、インメモリ・アーキテクチャにより、ディスクキャッシュの読み書きは行わず、1 億個のレコードがすべてメモリ内に収まるようになります。次に、大規模なデータセットを使用して、ディスク上にあるデータの大部分を各データベースがどのように実行するかをテストしました。 4 億(400M) レコードのデータセットでは、データの約 70%がディスクに配置されます。いずれのデータベースでも、二次ストレージの使用を最小限に抑えるため、圧縮を使用するように設定しています。Java がヒープ空間を使い果たすのを防ぐため、すべてのワークロード構成ファイルで「predefinetagstoreused」フィールドを false に設定しました。

以下で YCSB-TS ベンチマークテストの実行は、すべての YCSB-TS レコードがいずれかのデータベースに挿入されてから 1 時間後にしています。1 時間後としたのは、各データベースがすべてのハウスキーピング操作を完了させるまで待つことで、読み取りおよびスキャンのベンチマーク中の公平性を保つためです。

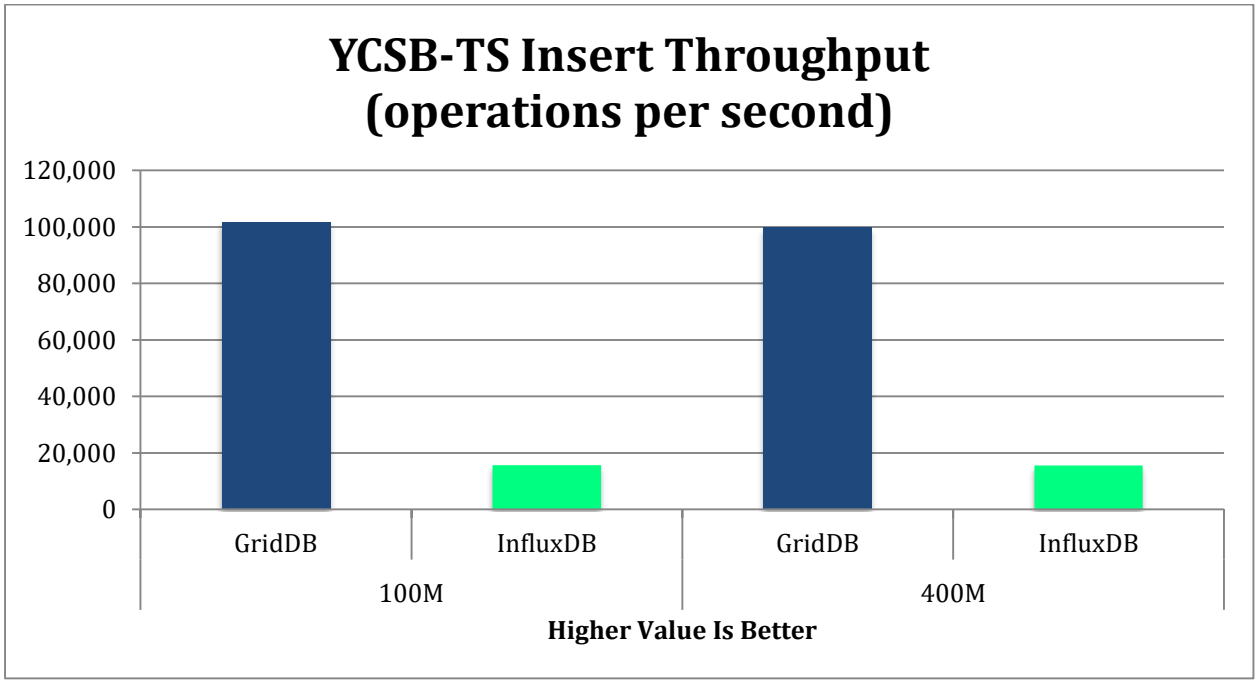

Figure 1: Insert Throughput

Read オペレーション

ワークロード A は、すべてのレコードがデータベースにロードされた後の読み取り操作のみで構成されたワークロード構成です。 各読み取り操作は、その行キーとして特定のタイムスタンプを持つレコードを検索します。 検索されるタイムスタンプは、YCSB-TS によってランダムに生成されます。

1 億(100M)レコードのテストでは、GridDB は InfluxDB に比べて約 8 倍の処理(Operations per second)を実行できました。より大きな 4 億(400M)レコードでは、GridDB のパフォーマンスは約 4 倍程度低下しましたが、InfluxDB のパフォーマンスは約 5 倍低下したため、結果として、GridDB は IncluxDB の 10 倍近くのパフォーマンスを持つことがわかりました。

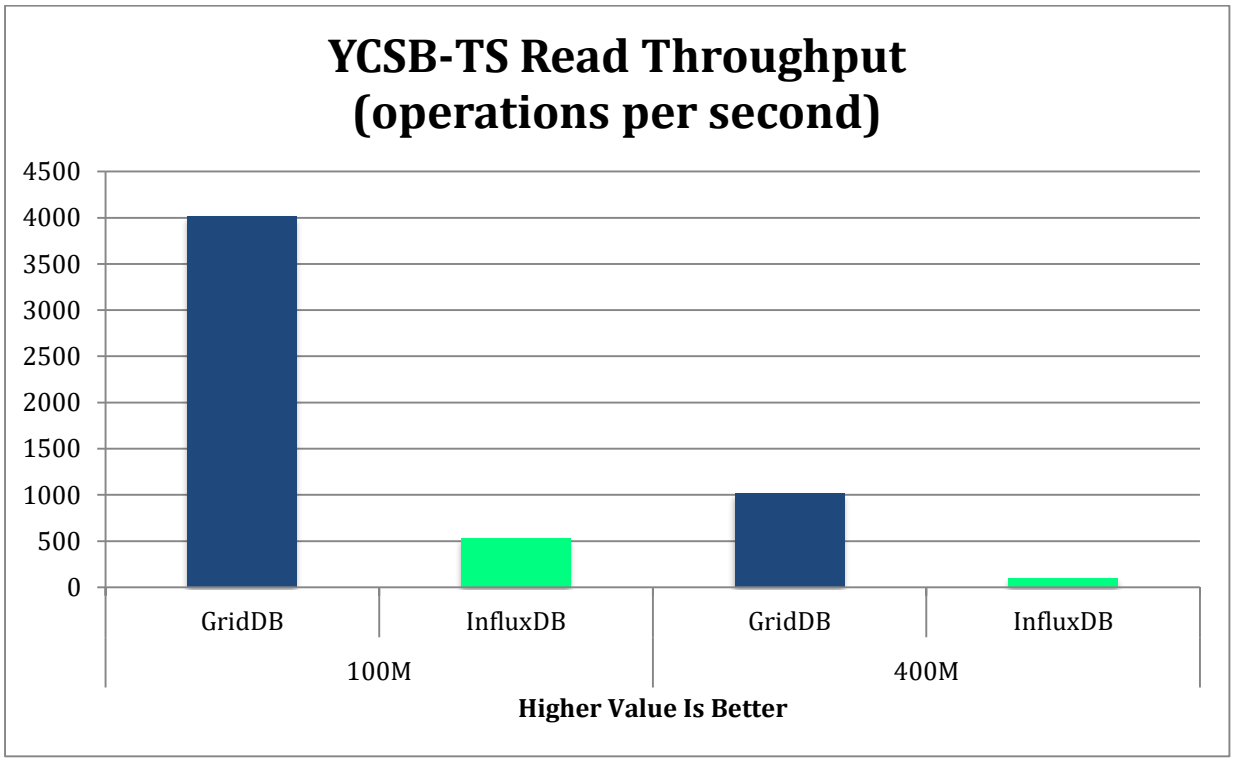

Figure 2: Read Throughput.

Scan オペレーション

ワークロード B は、「スキャン操作」 100%で構成されるワークロードです。スキャン操作の詳細は、25%が一般的なスキャン検索（Scan searches）、25%はカウント操作（Count operations）、25%は合計操作（Sum operations）、25%は平均操作（Average operation） となります。

スキャン検索は、2 つのランダムに生成されたタイムスタンプ値の間の行を検索します。 カウント操作は、2 つのタイム・スタンプ値の間にある時系列コンテナ内の行数を数えます。 合計操作、平均操作は、タ

イムスタンプ間にある各行の値の合計または平均を計算します。クエリ範囲として使用されるこれらのタイムスタンプ値は常に、ワークロード構成ファイルで指定された insertstart フィールドと insertend フィールドの間にあります。

YCSB-TS は、実行中のワークロード全体のスループットと、スキャン検索、カウント操作、合計操作、平均操作の各レイテンシを報告します。

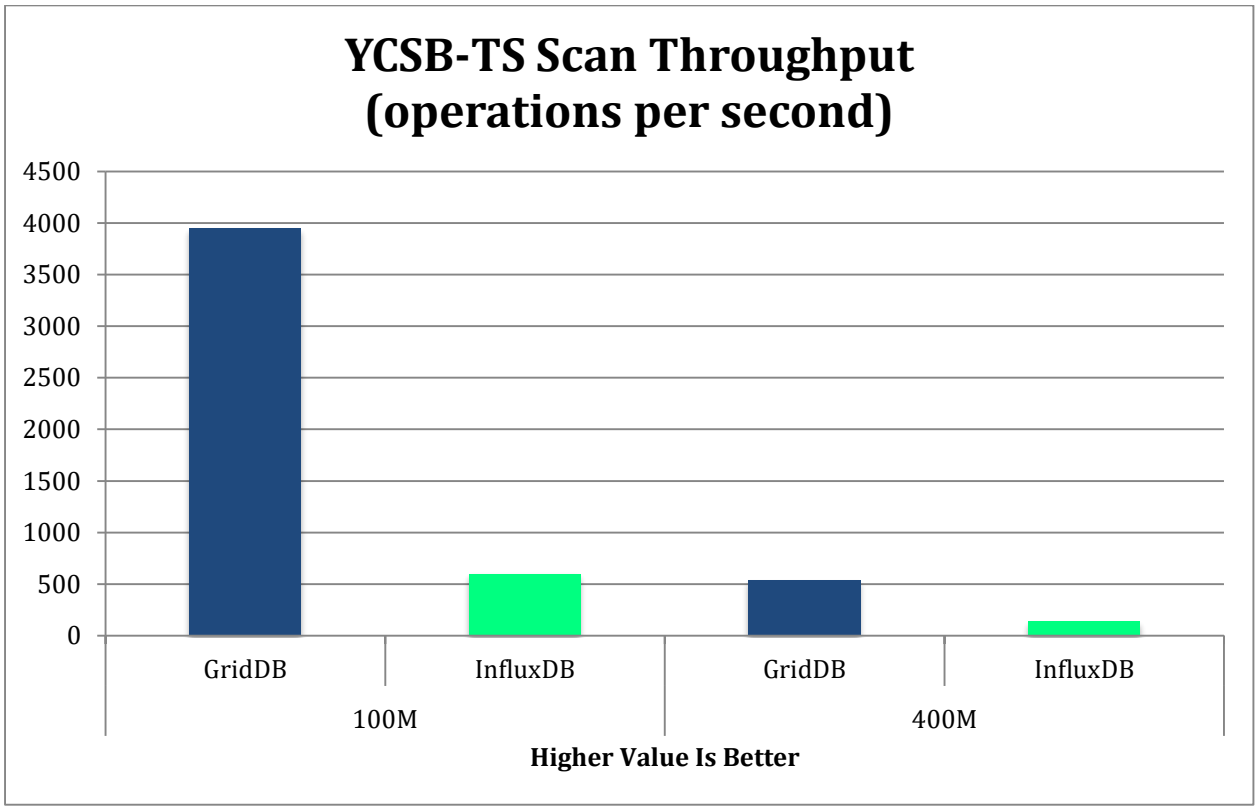

Figure 3: Total Scan/Aggregation Throughput.

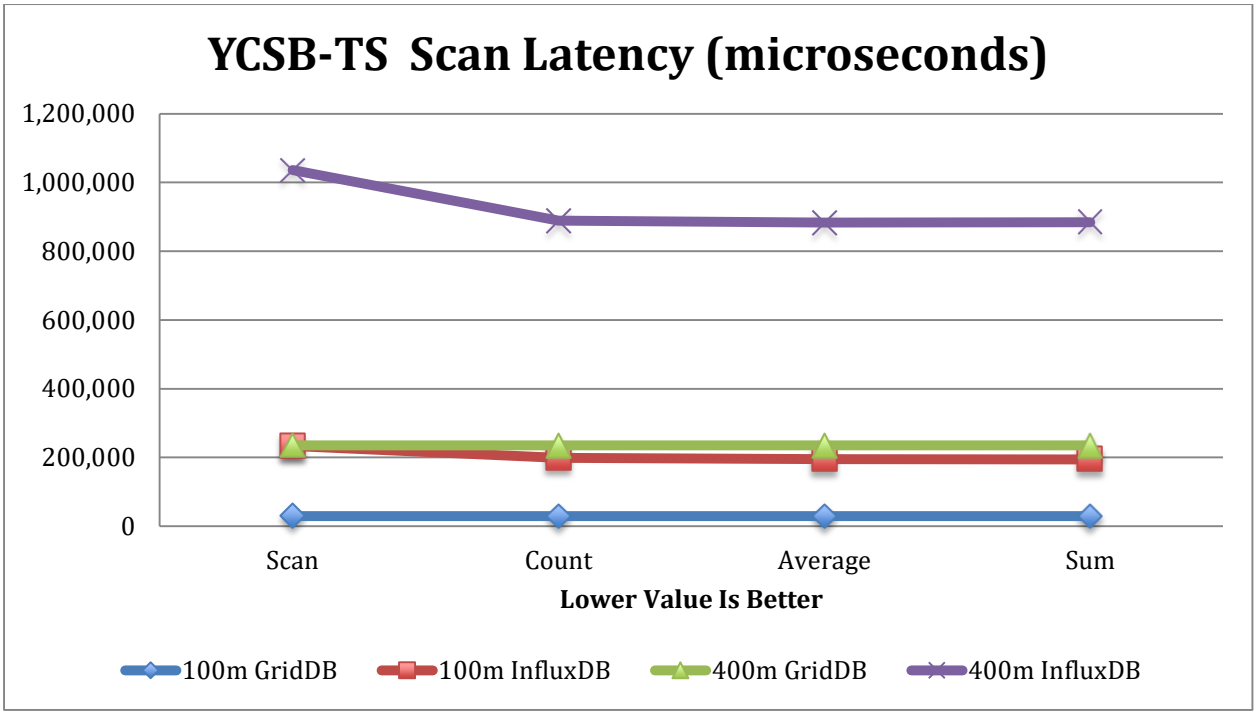

Figure 4: Scan, Count, Average, and Sum Latencies.

Database サイズ

GridDB と InfluxDB のサイズ効率を比較するために、ディスク上のデータのサイズを、ロードが完了した直後と、Write-Ahead-Log ファイルがフラッシュされた後に測定しました。

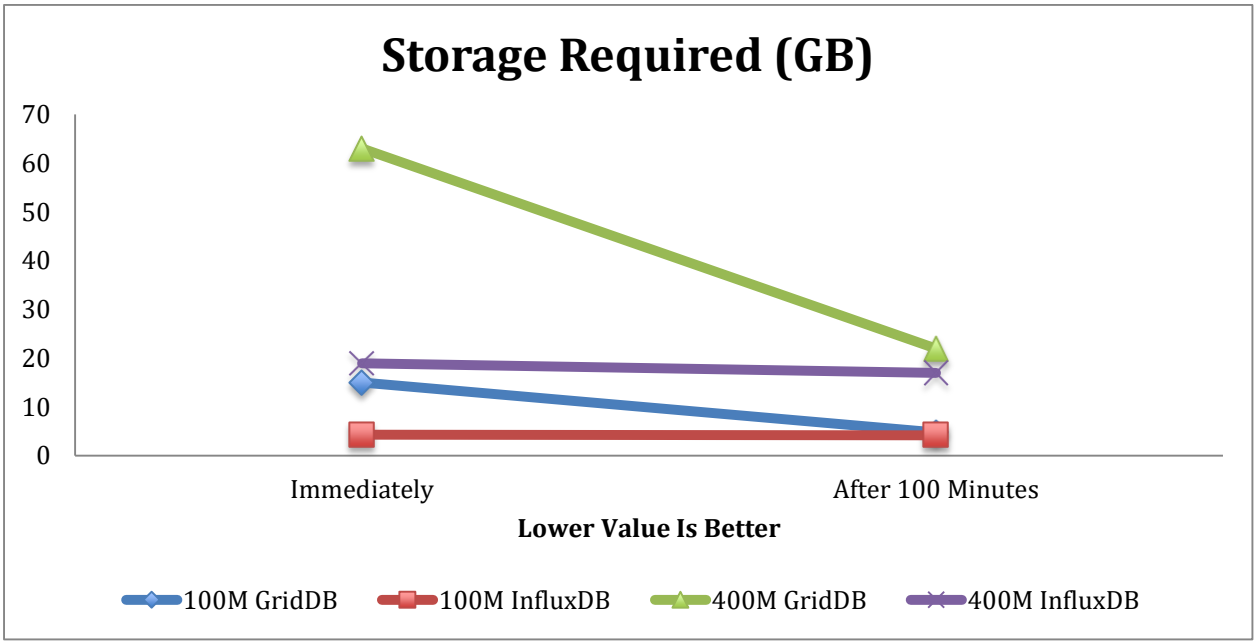

Figure 5: Storage Requirements

表形式での結果

以下は、すべての測定結果を表形式で示したものです。スループット測定値はすべて「1 秒当たりの操作数(Operations per second)」単位で、待ち時間測定値は「マイクロ秒」単位です。

Throughput (operations per second)

Test	Size	GridDB	InfluxDB	Advantage
Load	100M	101,793.7	15,637.3	GridDB 651% Better
	400M	99,771.1	15,512.0	GridDB 643% Better
Workload A	100M	4,013.3	525.6	GridDB 764% Better
	400M	1,014.6	102.4	GridDB 991% Better
Workload B	100M	3,948.0	599.6	GridDB 658% Better
	400M	535.8	136.3	GridDB 393% Better

Table 3: Workload Throughputs

Latencies (microseconds)

Operation	Size	GridDB	InfluxDB	Advantage
Load	100M	1,244.4	8,178.7	GridDB 657% better
	400M	1,250.1	8,246.3	GridDB 660% better
Read	100M	28,681.6	236,671.5	GridDB 825% better
	400M	122,946.4	1,141,034.2	GridDB 928% better
Scan	100M	29,711.5	233,252.0	GridDB 785% better
	400M	235,213.9	1,035,776.4	GridDB 440% better
Count	100M	29,130.8	199,063.9	GridDB 683% better
	400M	234,780.9	888,515.0	GridDB 378% better
Average	100M	29,082.4	195,110.7	GridDB 671% better
	400M	234,858.1	882,958.3	GridDB 376% better
Sum	100M	29,068.2	194,563.9	GridDB 669% better
	400M	234,796.2	884,338.7	GridDB 377% better

Table 4: Operation Latencies

Data Storage Size

		0 minutes	100 minutes
100M	GridDB	15GB	4.8GB
	InfluxDB	4.3GB	4.2GB
400M	GridDB	62GB	21GB
	InfluxDB	19GB	17GB

Table 5: Storage Requirements

CPU and Memory Usage

Test	Workload Size	Measurement	GridDB	InfluxDB
Load	100M	CPU Usage	317%	400%
		Memory Usage	5.5 GB	2.7GB
	400M	CPU Usage	305%	405%
		Memory Usage	5.8 GB	10.2GB
Workload A	100M	CPU Usage	288%	735%
		Memory Usage	5.5GB	4.1GB
	400M	CPU Usage	49.6%	722.5%
		Memory Usage	5.6GB	10.4GB
Workload B	100M	CPU Usage	231%	743%
		Memory Usage	5.5GB	5.2GB
	400M	CPU Usage	61.4%	713%
		Memory Usage	5.8GB	10.3GB

Table 6: CPU/Memory Usage.

結論

GridDB のインメモリ指向のハイブリッドストレージアーキテクチャは、InfluxDB と比較してより高いパフォーマンスを提供することが示されました。 GridDB は、データサイズがメモリに収まる範囲でも、メモリに収まりきらないような大きい場合でも、常に強力なパフォーマンスを発揮し、InfluxDB よりも高いスループットと低いレイテンシを維持することができました。 GridDB はまた、個々の時系列コンテナのデータが増加しても、一貫性と信頼性の高いパフォーマンスを維持します。つまり、大きなデータセットを取り扱う場合でも、多数のコンテナに分散配置する必要はありません。 これにより GridDB は、より長い範囲のより多くのタイムスタンプを格納することができます。

以上のテスト結果は、増大するデータセットや多数のハードウェア構成に対応する GridDB の優れた能力を示しています。 GridDB の示した高スループットと低レイテンシは、GridDB が、よりスケーラブルで柔軟な時系列データベース（TSDB）である証明と言えるでしょう。

Appendices

gs_node.json


```
{
	"dataStore":{
		"dbPath":"data",
		"storeMemoryLimit":"6192MB",
		"storeWarmStart":true,
		"storeCompressionMode": "COMPRESSION",
		"concurrency":8,
		"logWriteMode":1,
		"persistencyMode":"NORMAL",
		"affinityGroupSize":4
	},
	"checkpoint":{
		"checkpointInterval":"1200s",
		"checkpointMemoryLimit":"1024MB",
		"useParallelMode":false
	},
	"cluster":{
		"servicePort":10010
	},
	"sync":{
		"servicePort":10020
	},
	"system":{
		"servicePort":10040,
		"eventLogPath":"log"
	},
	"transaction":{
		"servicePort":10001,
		"connectionLimit":5000
	},
	"trace":{
		"default":"LEVEL_ERROR",
		"dataStore":"LEVEL_ERROR",
		"collection":"LEVEL_ERROR",
		"timeSeries":"LEVEL_ERROR",
		"chunkManager":"LEVEL_ERROR",
		"objectManager":"LEVEL_ERROR",
		"checkpointFile":"LEVEL_ERROR",
		"checkpointService":"LEVEL_INFO",
		"logManager":"LEVEL_WARNING",
		"clusterService":"LEVEL_ERROR",
		"syncService":"LEVEL_ERROR",
		"systemService":"LEVEL_INFO",
		"transactionManager":"LEVEL_ERROR",
		"transactionService":"LEVEL_ERROR",
		"transactionTimeout":"LEVEL_WARNING",
		"triggerService":"LEVEL_ERROR",
		"sessionTimeout":"LEVEL_WARNING",
		"replicationTimeout":"LEVEL_WARNING",
		"recoveryManager":"LEVEL_INFO",
		"eventEngine":"LEVEL_WARNING",
		"clusterOperation":"LEVEL_INFO",
		"ioMonitor":"LEVEL_WARNING"
```

```
        }
}
```

gs_cluster.json

```
{
        "dataStore":{
                "partitionNum":128,
                "storeBlockSize":"64KB"
        },
        "cluster":{
                "clusterName":"defaultCluster",
                "replicationNum":2,
                "notificationAddress":"239.0.0.1",
                "notificationPort":20000,
                "notificationInterval":"5s",
                "heartbeatInterval":"5s",
                "loadbalanceCheckInterval":"180s"
        },
        "sync":{
                "timeoutInterval":"30s"
        },
        "transaction":{
                "notificationAddress":"239.0.0.1",
                "notificationPort":31999,
                "notificationInterval":"5s",
                "replicationMode":0,
                "replicationTimeoutInterval":"10s"
        }
}
```

influxdb.conf

```
[meta]
  # Where the metadata/raft database is stored
  dir = "/var/lib/influxdb/meta"

  # Automatically create a default retention policy when creating a database.
  # retention-autocreate = true

  # If log messages are printed for the meta service
  # logging-enabled = true
[data]
  # The directory where the TSM storage engine stores TSM files.
  dir = "/var/lib/influxdb/data"

  # The directory where the TSM storage engine stores WAL files.
  wal-dir = "/var/lib/influxdb/wal"
  # wal-fsync-delay = "0s"
  index-version = "tsi1"
  trace-logging-enabled=true
  # CacheMaxMemorySize is the maximum size a shard's cache can
  # reach before it starts rejecting writes.
  cache-max-memory-size = 1048576000

  # CacheSnapshotMemorySize is the size at which the engine will
  # snapshot the cache and write it to a TSM file, freeing up memory
  cache-snapshot-memory-size = 26214400

  cache-snapshot-write-cold-duration = "10s"
```

```
# compact-full-write-cold-duration = "4h"
  # max-concurrent-compactions = 0
  # The maximum series allowed per database before writes are dropped.  This limit can prevent
  # high cardinality issues at the database level.  This limit can be disabled by setting it to
  # 0.
  max-series-per-database = 0

  # The maximum number of tag values per tag that are allowed before writes are dropped.  This limit
  # can prevent high cardinality tag values from being written to a measurement.  This limit can be
  # disabled by setting it to 0.
  max-values-per-tag = 0

[coordinator]
  # The default time a write request will wait until a "timeout" error is returned to the caller.
  # write-timeout = "10s"

  # The maximum number of concurrent queries allowed to be executing at one time.  If a query is
  # executed and exceeds this limit, an error is returned to the caller.  This limit can be disabled
  # by setting it to 0.
  max-concurrent-queries = 128

  # The maximum time a query will is allowed to execute before being killed by the system.  This limit
  # can help prevent run away queries.  Setting the value to 0 disables the limit.
  # query-timeout = "0s"

  # The time threshold when a query will be logged as a slow query.  This limit can be set to help
  # discover slow or resource intensive queries.  Setting the value to 0 disables the slow query logging.
  # log-queries-after = "0s"

  # The maximum number of points a SELECT can process.  A value of 0 will make
  # the maximum point count unlimited.  This will only be checked every 10 seconds so queries will not
  # be aborted immediately when hitting the limit.
  max-select-point = 0

  # The maximum number of series a SELECT can run.  A value of 0 will make the maximum series
  # count unlimited.
  max-select-series = 0

  # The maximum number of group by time bucket a SELECT can create.  A value of zero will max the maximum
  # number of buckets unlimited.
  # max-select-buckets = 0

###
### [retention]
###
### Controls the enforcement of retention policies for evicting old data.
###

[retention]
  # Determines whether retention policy enforcement enabled.
  # enabled = true

  # The interval of time when retention policy enforcement checks run.
  # check-interval = "30m"
[shard-precreation]
  # Determines whether shard pre-creation service is enabled.
  # enabled = true

  # The interval of time when the check to pre-create new shards runs.
  # check-interval = "10m"

  # The default period ahead of the endtime of a shard group that its successor
  # group is created.
  # advance-period = "30m"
```

```
[monitor]
  # Whether to record statistics internally.
  # store-enabled = true

  # The destination database for recorded statistics
  # store-database = "_internal"

  # The interval at which to record statistics
  # store-interval = "10s"

[http]
  # Determines whether HTTP endpoint is enabled.
  # enabled = true

  # The bind address used by the HTTP service.
  # bind-address = ":8086"

  # Determines whether user authentication is enabled over HTTP/HTTPS.
  # auth-enabled = false

  # The default realm sent back when issuing a basic auth challenge.
  # realm = "InfluxDB"

  # Determines whether HTTP request logging is enabled.
  log-enabled = false

  # Determines whether detailed write logging is enabled.
  write-tracing = false

  # Determines whether the pprof endpoint is enabled.  This endpoint is used for
  # troubleshooting and monitoring.
  # pprof-enabled = true

  # Determines whether HTTPS is enabled.
  # https-enabled = false

  # The SSL certificate to use when HTTPS is enabled.
  # https-certificate = "/etc/ssl/influxdb.pem"

  # Use a separate private key location.
  # https-private-key = ""

  # The JWT auth shared secret to validate requests using JSON web tokens.
  # shared-secret = ""

  # The default chunk size for result sets that should be chunked.
  # max-row-limit = 0

  # The maximum number of HTTP connections that may be open at once.  New connections that
  # would exceed this limit are dropped.  Setting this value to 0 disables the limit.
  # max-connection-limit = 0

  # Enable http service over unix domain socket
  # unix-socket-enabled = false

  # The path of the unix domain socket.
  # bind-socket = "/var/run/influxdb.sock"
[subscriber]
  # Determines whether the subscriber service is enabled.
  # enabled = true

  # The default timeout for HTTP writes to subscribers.
  # http-timeout = "30s"

  # Allows insecure HTTPS connections to subscribers.  This is useful when testing with self-
  # signed certificates.
```

```
  # insecure-skip-verify = false

  # The path to the PEM encoded CA certs file. If the empty string, the default system certs will be used
  # ca-certs = ""

  # The number of writer goroutines processing the write channel.
  # write-concurrency = 40

  # The number of in-flight writes buffered in the write channel.
  # write-buffer-size = 1000
…..
[continuous_queries]
  # Determines whether the continuous query service is enabled.
  # enabled = true

  # Controls whether queries are logged when executed by the CQ service.
  # log-enabled = true

  # interval for how often continuous queries will be checked if they need to run
  # run-interval = "1s"
```